Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

ICE: Uma solução geral para a travessia de NAT

José Henrique de Oliveira Varanda

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientador
Prof. Ms. João José Costa Gondim

Brasília
2008

Universidade de Brasília – UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Curso de Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Guilherme Albuquerque Pinto

Banca examinadora composta por:

Prof. Ms. João José Costa Gondim (Orientador) – CIC/Unb
Profª. Drª. Priscila Sólis Barreto – CIC/Unb
Prof. Dr. Jacir Luiz Bordim – CIC/Unb

CIP – Catalogação Internacional na Publicação

Varanda, José Henrique de Oliveira.

ICE: Uma solução geral para a travessia de NAT / José Henrique de Oliveira Varanda. Brasília : UnB, 2008.
92 p. : il. ; 29,5 cm.

Monografia (Graduação) – Universidade de Brasília, Brasília, 2008.

1. ICE, 2. Conectividade, 3. STUN, 4. TURN, 5. NAT, 6. IETF, 7. Travessia de NAT, 8. SIP, 9. SDP, 10. UDP, 11. IPv4, 12. IPv6, 13. Exaustão espaço endereçamento

CDU 004

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro – Asa Norte
CEP 70910–900
Brasília – DF – Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

ICE: Uma solução geral para a travessia de NAT

José Henrique de Oliveira Varanda

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Ms. João José Costa Gondim (Orientador)
CIC/Unb

Profª Drª Priscila Sólis Barreto
CIC/Unb

Prof. Dr. Jacir Luiz Bordim
CIC/Unb

Prof. Dr. Guilherme Albuquerque Pinto
Coordenador do Bacharelado em Ciência da Computação

Brasília, 3 de julho de 2008

Dedicatória

Dedico este trabalho a meus pais e irmãos pelas palavras de apoio e confiança e, especialmente, a Lorena por me manter motivado e focado sempre com palavras de amor e carinho ao longo desta jornada. Sem vocês este trabalho não seria possível.

Agradecimentos

Agradeço ao Prof. Dr. João José Costa Gondim por acreditar neste trabalho e viabilizar a sua realização nos momentos mais difíceis. Sua atenção e cobrança foram fundamentais.

Aos demais professores e funcionários da UnB, especialmente do CiC, por oferecerem uma experiência única de aprendizado e relacionamento que certamente estará presente ao longo de minha vida.

Aos colegas da SOCIA, em especial ao André, pelo apoio e compreensão e por permanecerem firmes e focados em suas atividades mesmo durante minha ausência.

Resumo

Este trabalho apresenta a questão da exaustão do espaço de endereçamento de endereços IPv4 e mostra como ela está relacionada a popularidade do NAT. Foi feita uma análise dos problemas causados pelo amplo uso do NAT, avaliando-se o seu impacto na conectividade de nós em redes privadas e públicas.

Os principais métodos para realização da travessia de NAT foram estudados, sendo que o foco principal é o protocolo ICE. Este protocolo é um padrão proposto pelo IETF para o estabelecimento de conexões interativas entre agentes baseado no modelo de Oferta e Resposta compreendendo conexões ponto a ponto e travessia de NAT.

Foi desenvolvida uma biblioteca em Java chamada JICE que implementa este padrão. Ela foi implementada com a biblioteca Apache MINA proporcionando o uso de boas práticas de programação em rede com foco em performance para esta plataforma. As principais questões encontradas utilizando-se estas ferramentas foram apresentadas incluindo seus benefícios e desvantagens.

Testes de carga e conectividade foram realizados em um ambiente controlado baseado em máquinas virtuais. Este ambiente foi usado para simular redes e cenários de NAT. Foi realizada uma avaliação de pacotes de rede para verificar a aderência do JICE ao ICE e seu grau de compatibilidade comparado com outra implementação do ICE.

Demonstrou-se que este protocolo representa um forma eficiente de se realizar a travessia de NAT em nível de aplicação cliente na maioria dos cenários avaliados. Isto o posiciona como uma solução viável para o problema de travessia de NAT.

Palavras-chave: ICE, Conectividade, STUN, TURN, NAT, IETF, Travessia de NAT, SIP, SDP, UDP, IPv4, IPv6, Exaustão espaço endereçamento

Abstract

This work presents the IPv4 address exhaustion issue and shows how it's related to NAT popularity. An analysis of the problems caused by the widespread use of NAT was conduted, evaluating it's impact on nodes in private and public networks.

The main NAT Traversal protocols and techniques were studied and presented focusing on the ICE protocol. This protocolol is a IETF's proposed standard for the interactive connectivity estabilishiment among agents based on the Offer/Answer model comprehending peer to peer connections and NAT Traversal.

A Java library named JICE was created to implement this IETF's standard. It was implemented using the Apache MINA library in order to use network programming best practices focused in performance on this platform. The main issues found while working with these tools were reported including it's benefits and disadvantages.

Load and connectivity tests were performed in a controlled environment based on virtual machines. This environment was used to simulate networks and NAT scenarios, among them multi-homed and dual-stack client scenarios. A network packages evaluation was performed in order to check JICE's adherence to ICE and it's compatibility level compared to another ICE implementation.

It was demonstrated that this protocol represents an efficient way to execute the NAT Traversal in a client application level in the majority of the scenarios evaluated. This positions it as a viable solution to the NAT Traversal problem.

Keywords: ICE, STUN, TURN, Connectivity, NAT, IETF, NAT Traversal, SIP, SDP, UDP, IPv4, IPv6, IP address exhaustion

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Visão Geral

1.1 Introdução

Durante os últimos anos a internet tem crescido em ritmo acelerado. Além disso, a convergência de redes e tecnologias analógicas para meios digitais, tais como telefones e televisões têm gerado uma pressão ainda maior nesta tendência, aumentando exponencialmente o número de nós conectados a internet.

Neste cenário está a questão da conectividade, em especial, a infra-estrutura atual de endereçamento destes nós, o chamado IPv4. Este padrão de endereçamento já encontra limitações e não consegue atender a todos que desejam se conectar a grande rede. Existem mais clientes do que este protocolo consegue endereçar e este é um problema imediato que as redes já estão combatendo.

Como principal solução foi desenvolvido um conjunto de técnicas chamadas Network Address Translation (NAT) [1]. Ele consegue resolver o problema de endereçamento através do compartilhamento de um endereço externo por diversos nós de uma determinada rede interna ao NAT. Ou seja, o endereço externo é o ponto de ligação entre a rede interna e a rede externa, comportando-se como uma ponte de endereçamento.

Ao fazer isso, o NAT contribui ainda para segurança das redes privadas, pois, ao mapear o endereçamento das sessões de comunicação entre os nós internos e os externos, ele impede que sejam iniciadas novas sessões a partir dos nós externos. Ou seja, a requisição para o início de novas sessões de comunicação pode ser realizada apenas por nós internos. Pois assim, pode-se realizar o mapeamento adequado de qual sessão pertence a qual nó interno.

1.2 Motivação para as soluções de travessia de NAT

Agora pensando-se no cenário onde uma requisição é enviada inicialmente por um nó externo, o NAT não tem como identificar, a princípio, para qual nó na rede interna ele deverá encaminhar esta requisição. Nestes casos, o NAT, em geral, descarta requisições externas silenciosamente, inviabilizando o estabelecimento da comunicação neste sentido. Além disso, ele não permite identificar os motivos desta falha, se ela se deve a um NAT ou não, ou até mesmo a identificação das características e topologias dos n níveis de NAT possíveis entre os nós.

Logo, o NAT quebra o modelo ponto a ponto inicial do IPv4 o que impede que nós que estejam entre dois ou mais NAT distintos iniciem uma sessão de comunicação. Necessidade esta que é crescente devido às novas aplicações emergindo na internet tais como sistemas descentralizados, sistemas de VoIP, sistemas P2P, aplicações cooperativas e colaborativas, comunicação pessoal, transmissões em tempo real, jogos em rede, sistemas de controle, compartilhamento de recursos, entre outros.

Para transpor esta dificuldade criada pelo NAT foram desenvolvidas diversas técnicas classificadas como *NAT Traversal*. Estas técnicas permitem que nós internos a NAT distintos consigam estabelecer um canal de comunicação direto, ou no pior dos casos, uma canal de comunicação indireto através de um servidor acessível por ambos os nós.

Entre estas técnicas foi dado ênfase ao ICE [12] que é um padrão proposto pelo IETF para melhorar a conectividade entre agentes comunicando fluxos multimída através de uma modelo de Oferta e Resposta, como ocorre nos cenários de uso do VoIP ou *streaming* de vídeo, oferecendo principalmente recursos para a travessia de NAT.

1.3 Objetivos

O objetivo deste trabalho é realizar a implementação do protocolo ICE e avaliar sua eficácia em termos de conectividade e sua eficiência em resolver o problema de conectividade trazido pelo NAT em termos quantitativos comparativamente as soluções existentes.

Para isso, ele busca realizar um estudo sobre NAT, mostrar como a questão da exaustão do espaço de endereçamento promoveu sua popularização, avaliar seus princípios de funcionamento e apresentar como ele se classifica em diversos tipos de acordo com o nível de conectividade disponível a seus clientes.

Ele prevê ainda, um levantamento das principais técnicas que permitem a travessia de NAT apresentando suas abordagens, vantagens e desvantagens, re-

alizando um comparativo entre as mesmas. Neste momento, é dado enfoque ao ICE e como este padrão oferece uma solução geral, padronizada e de simples implantação para esta questão em nível cliente.

Isso envolve um estudo detalhado deste protocolo, assim como dos protocolos associados ou estendidos pelo mesmo. Esta análise apresenta suas interdependências, seu ciclo de vida e estrutura de pacotes e mensagens de rede, de forma a viabilizar sua implementação.

1.3.1 Objetivos específicos

Implementar o protocolo ICE para a plataforma Java. Neste sentido, avaliar os recursos e ferramentas para a programação de rede, envolvendo servidores e clientes, nesta plataforma. Especificamente, avaliar o modelo não blocante e assíncrono de acesso a rede e de como a ferramenta Apache MINA [20] contribui para simplificar, padronizar e otimizar a utilização destes recursos.

Realizar testes de diferentes cenários de uso de NAT em ambiente controlado. Especificamente, utilizar máquinas virtuais para simular os diversos agentes e servidores envolvidos nos cenários onde o ICE poderá ser utilizado. Avaliar a eficiência desta abordagem apresentando suas principais vantagens e desvantagens.

1.4 Estrutura do projeto

Este trabalho é uma monografia de graduação, logo, sua estrutura apresenta primeiro uma contextualização teórica e um estudo bibliográfico. Depois descreve a implementação do protocolo objeto de estudo, o ICE. Após isso, faz um estudo da solução desenvolvida envolvendo testes e avaliação de resultados. E por fim, apresenta suas conclusões e trabalhos futuros. Esta estrutura está compreendida nos seguintes capitulos:

- **Capítulo 1** - Nele encontra-se a introdução deste estudo onde é feita uma contextualização apresentando suas motivações e objetivos.
- **Capítulo 2** - Trata dos protocolos de internet com enfoque na sua infraestrutura de endereçamento abordando a questão da exaustão deste espaço bem como as soluções dadas a esta questão. O foco é mostrar que as soluções de travessia de NAT, embora paliativas, deverão estar presentes por algum tempo em diversos cenários.
- **Capítulo 3** - Apresenta o conceito de NAT em detalhes, sua classificação com relação a conectividade e as técnicas usadas para sua travessia.

- **Capítulo 4** - Descreve o protocolo ICE detalhando suas interações com outros protocolos, seu ciclo de vida e a estrutura de suas mensagens e pacotes.
- **Capítulo 5** - Aborda a implementação deste protocolo, descrevendo a plataforma e ferramentas utilizadas, sua arquitetura e os problemas encontrados durante o desenvolvimento.
- **Capítulo 6** - Relata os testes detalhando seu ambiente e metodologia fazendo uma análise dos resultados encontrados.
- **Capítulo 7** - Conclui relacionando os resultados obtidos com os objetivos estabelecidos e ainda sugere trabalhos futuros e linhas de pesquisa a serem explorados.

Capítulo 2

Protocolos de Internet

Este capítulo apresenta o protocolo de internet e suas versões dando ênfase a questão do espaço de endereçamento. Trata da questão da exaustão, suas soluções e seus impactos na questão da conectividade. O objetivo é mostrar como esta questão favoreceu a popularização do NAT criando a necessidade para o ICE.

2.1 Protocolo de Internet versão 4

O protocolo de internet (Internet Protocol - IP) [18] é um protocolo orientado a dados para a comunicação de dados em uma rede de pacotes comutados. Entre outros serviços, ele determina o endereçamento atribuído a nós de uma determinada rede de forma a identificá-lo unicamente para o envio e recebimento de pacotes.

O IPv4 é a quarta geração do protocolo de internet e é a primeira versão a ser amplamente utilizada. O IPv4 é o protocolo de camada de rede, segundo o modelo OSI, dominante e é o padrão amplamente utilizado na internet.

Ele é descrito segundo o RFC 791 do IETF e foi publicado em 1981, o que tornou obsoleto seu antecessor definido no RFC 760 de janeiro de 1980. Seu principal propósito é prover endereçamento único e global para assegurar que dois computadores, ao se comunicarem, possam se identificar unicamente.

O IPv4 é um protocolo orientado a dados para ser usado em redes de comutação de pacotes, tais como o padrão Ethernet. Ele busca um melhor esforço para a entrega dos pacotes, porém, não a garante. Assim como, não assegura a integridade dos dados transmitidos, ou seja, o transporte de pacotes através do IPv4 pode resultar em duplicações ou ainda em recebimentos desordenados. Normalmente essas questões são abordadas por protocolos das camadas superiores tais como o TCP e em parte pelo UDP [21].

2.1.1 Endereçamento

O IPv4 utiliza endereços de 32 bits (4 bytes) com pode ser visto na figura 2.1, o que limita o seu espaço de endereçamento em 4.294.967.296 (2^{32}) possíveis endereços únicos. Entretanto, alguns desses endereços foram reservados para propósitos especiais tais como redes privativas (18 milhões de endereços) ou endereços para *multicast* (1 milhão de endereços).

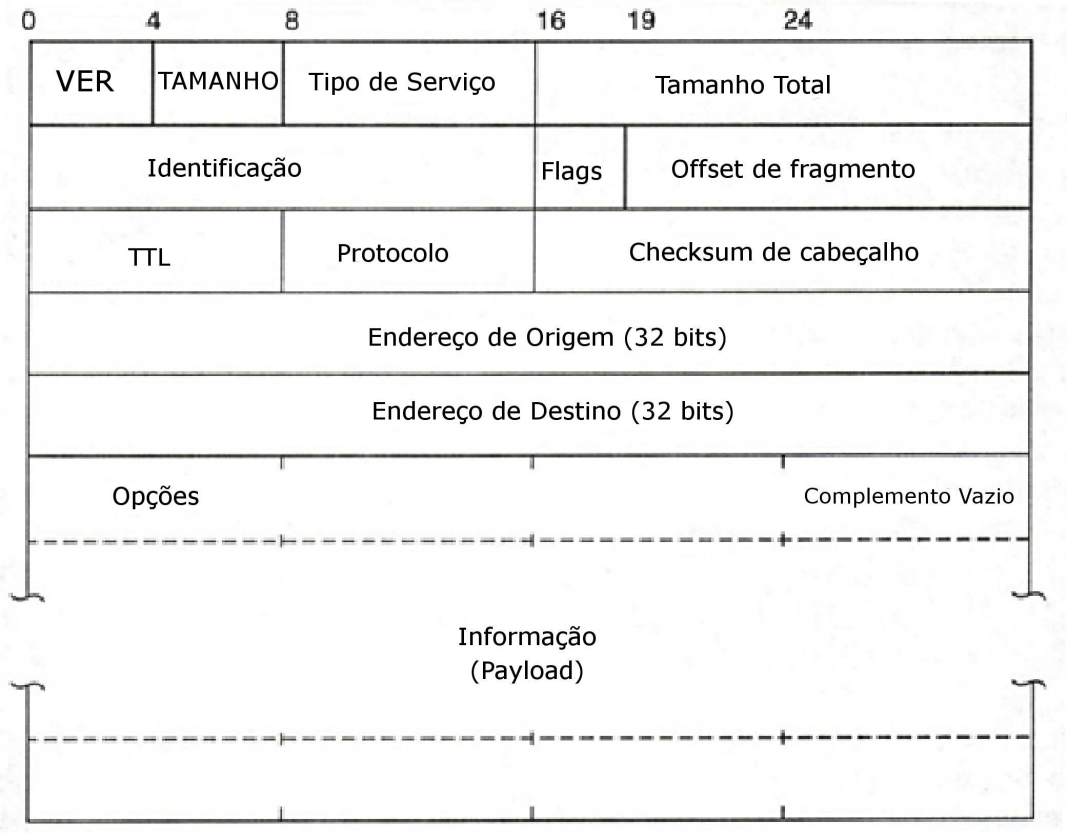


Figura 2.1: Estrutura de pacote e endereçamento do IPv4.

Dos mais de 4 bilhões de endereços permitidos através do IPv4, apenas 3 intervalos de endereços são reservados para o uso em redes privadas [2]. Esses intervalos não são roteáveis fora dessas redes, logo nós nessas redes não podem se comunicar com as redes públicas diretamente. Porém, eles podem realizar essa comunicação através da tradução de endereços de rede ou *Network Address Translation* (NAT).

A distribuição destes endereços é coordenada pela a IANA [26] e delegada aos registros regionais de internet, assim como o Comitê Gestor da Internet no Brasil (CGI.br). Estes registros distribuem os endereços disponibilizados pelo IANA às companhias e provedores de acesso a internet em sua região. A coordenação para a alocação destes endereços aos registros regionais é feita no sentido de disponibilizar um estoque de endereços com duração prevista entre 12 e 18 meses entre cada requisição ao IANA. Isto faz com que os estoques tenham uma previsibili-

dade de duração equivalente em todas as regiões.

2.1.2 Exaustão

À medida que os endereços disponíveis vão sendo consumidos uma exaustão destes parece inevitável. O problema da escassez decorre do fato de cada vez mais existirem dispositivos conectados a internet, tais como computadores, assistentes digitais, celulares, modems, roteadores, pontos de acesso sem fio, telefones de voz sobre IP, video games, entre outros. Isto está representado no número crescente de pessoas que têm tido acesso a grande rede através de políticas de incentivo governamentais, da diminuição dos custos de acesso à banda larga, do crescimento econômico mundial experimentado na última década, enfim da recente popularização da internet.

Outra questão que agrava este quadro é a ineficiência no uso dos endereços disponíveis por parte de algumas organizações, que possuem blocos de endereços bem maiores do que elas de fato necessitam. Esses grandes blocos foram adquiridos principalmente durante a década de 80 e início da década de 90, quando ainda não havia muita preocupação sobre esta questão. Por exemplo, algumas grandes empresas e universidades receberam os chamados blocos /8 que possuem 16 milhões de endereço cada um. Além disso, muitas organizações ainda utilizam endereços IP públicos para dispositivos que não são acessados de fora de sua rede privativa e poderiam certamente estar endereçados internamente através de soluções tais como o NAT.

Além disso, algumas ineficiências causadas pelo uso de sub-redes tornam ainda mais difícil se aproveitar todos os endereços disponíveis em um determinado bloco de endereçamento, pois nem todos os endereços disponíveis para uma organização fazem parte de uma mesma sub-rede, tendo sua utilização preterida.

2.1.3 Mitigação

Para combater a questão da exaustão de endereços disponíveis foram desenvolvidas algumas técnicas que são complementares. Entre elas podemos destacar:

- **Uso de redes privadas** - Permite a reutilização das faixas de endereços de redes privadas entre organizações. Antes da popularização da internet era comum existirem redes privadas isoladas, logo que não possuíam conectividade com a internet. Assim, as redes privadas são o principal meio de combate ao problema da exaustão;
- **NAT** - Viabiliza a comunicação dos nós em redes privadas com nós nas redes públicas. Ou seja, favorece o compartilhamento de um endereço público entre vários nós de uma rede privada. O NAT se popularizou com o aumento de nós conectados a internet, já que é o principal elo de comunicação entre

redes privadas e públicas. Muitas das redes privadas pré-existentes vieram a se conectar a Internet através de NAT. Atualmente, inclusive as redes privadas dos provedores de acesso a internet também apresentam alguns níveis de NAT, o que favorece bastante a questão do endereçamento, apesar de dificultar a conectividade entre nós servidos pelas mesmas;

- **Dynamic Host Configuration Protocol (DHCP)** - Sua principal função é atribuir dinamicamente as configurações do protocolo IP para um nó. Seu principal benefício é favorecer a reutilização de endereços em uma determinada rede. É amplamente utilizado em redes privadas, pois viabiliza simples configuração e acesso a rede para os nós. Favorece o uso do NAT, já que torna simples a configuração do *gateway* para uso do mesmo;

- **Servidores virtuais baseados em nome** - Muito utilizado em servidores web, favorece a implantação de vários servidores em um mesmo endereço público. O chaveamento neste caso é feito pelo nome de cada servidor, ou do serviço sendo acessado. Por exemplo, um mesmo servidor Web pode prover mais de uma aplicação, de acordo com o nome do contexto sendo acessado. Além disso, pode se fazer o chaveamento através da porta acessada. Ou seja, existe um servidor diferente em cada porta. Em alguns casos, o servidor acessível na rede pública apenas encaminha a requisição para outros servidores em uma rede privada, de certa forma atuando como um NAT de alto nível e provendo ainda balanço de carga e resistência à falhas;

- **Maior controle dos registros regionais** - Com uma avaliação mais criteriosa na disponibilização dos endereços públicos espera-se evitar os exageros ocorridos no passado. Além disso, favorece o reaproveitamento daqueles endereços que não são mais utilizados;

- **Realocação de endereços** - Consiste em recuperar os enormes blocos de endereço disponibilizados para algumas organizações no início da internet e redistribuí-los de forma mais racional;

- **Adoção do IPv6** - A próxima versão do protocolo IP apresenta um espaço de endereçamento muito superior (2^{128} endereços), algumas ordens de grandeza, em relação ao padrão atual e é visto como a única solução definitiva para esta questão no médio e longo prazo.

2.1.4 Previsão para a data de exaustão

Algumas simulações detalhadas [27] realizadas por alguns registros regionais prevêem que a exaustão dos endereços não alocados em estoque no IANA ocorrerá entre janeiro e fevereiro de 2011. Essas previsões levam em consideração a demanda e a tendência atual em cada um dos registros regionais e não leva em consideração qualquer efeito que a própria exaustão possa causar na demanda, o que poderia adiantar estas previsões. Após a exaustão do estoque do IANA cada registro regional ainda terá estoque para alocar endereços por mais 10 a 18 meses aproximadamente, ou seja, a exaustão total de endereços disponíveis está prevista para

o fim de 2011 e início de 2012.

Assim, os registros regionais já estão aconselhando a comunidade que faz uso da internet a adequar suas redes ao protocolo IPv6 até o fim de 2010. Logo, o movimento de adoção e adequação das redes para este protocolo está ocorrendo em um ritmo crescente.

Por outro lado, estes estudos também demonstram que se os endereços já alocados e que não são devidamente aproveitados fossem recuperados e redistribuídos a alocação de endereços no padrão IPv4 poderá continuar até 2024. Esse é um dos fatores que levam a discussão se a mudança para o padrão IPv6 será rápida ou gradual.

2.2 Protocolo de Internet versão 6

O IPv6 é a sexta geração do protocolo de internet e é considerado o sucessor do IPv4 para uso geral na internet. A principal melhoria oferecida é o seu espaço de endereçamento, que é bem maior que o atual disponível. O seu endereço consiste em um número de 128 bits (16 bytes), que consiste em 2^{128} (aproximadamente 3.4 x 10^{38}) possíveis endereços, ou ainda 5x10^{28} endereços para cada um dos quase 6,5 bilhões de habitantes do planeta.

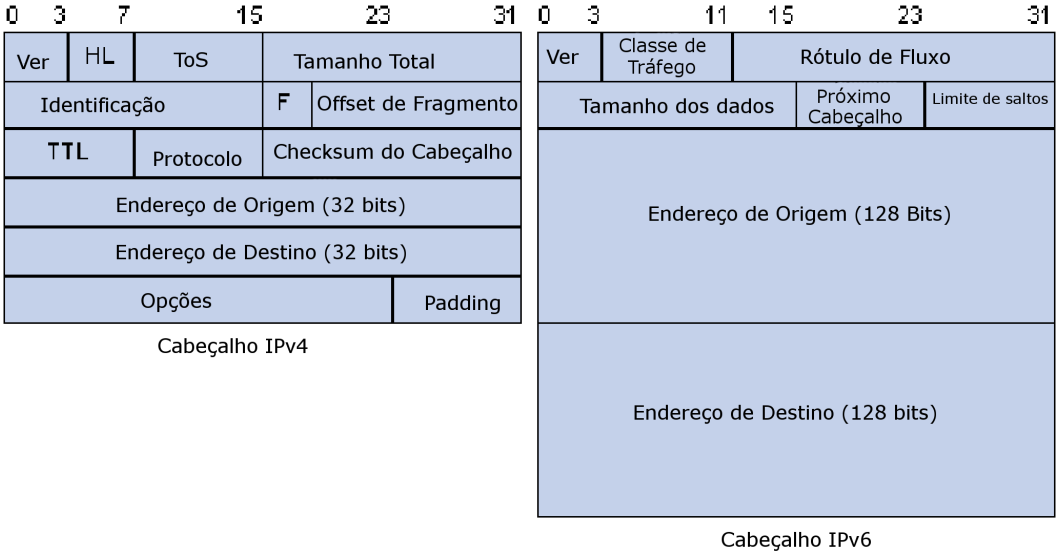


Figura 2.2: Comparação entre a estrutura de pacote do protocolo IPv4 e do IPv6.

O desenvolvimento do IPv6 se iniciou no início da década de 90, quando já estava claro que as mudanças feitas no IPv4 até então não seriam capazes de evitar a exaustão do espaço de endereçamento. No início de 1992 várias soluções foram propostas e ao final do ano o IETF anunciou o RFC 1650 intitulado "*IP, the Next Generation*"criando a área IPng para grupos de trabalho. Desde então,

vários grupos de trabalho foram criados para tratar do tema e em 1996 uma série de RFCs, começando com o RFC 2460, foram publicados definindo este protocolo.

Com exceção de algumas funcionalidades novas e restritas ao IPv6, a maioria das funcionalidades foi portada a partir do IPv4 sem maiores problemas. Na prática o IPv6 é uma extensão quase conservadora ao IPv4, pois muda essencialmente a estrutura de endereçamento. Logo, ele foi desenvolvido sabendo-se que conviveria por um bom tempo com o IPv4 durante o processo de transição. Esta abordagem permite que a maioria das alterações e atualizações seja feita via software com atualizações de drivers e sistemas operacionais em firmwares. Os atuais sistemas operacionais Windows Vista, Apple Mac OS X, Sun Solaris, IBM z/OS e Unix/Linux já possuem suporte IPv6 habilitado por padrão.

Além disso, as interfaces de redes dos computadores são, a princípio, compatíveis com o novo protocolo necessitando na maioria dos casos de algumas poucas atualizações de drivers. Com relação aos dispositivos de rede tais como roteadores e firewalls a adequação pode ser realizada através atualizações de seus sistemas operacionais em firmwares. Porém, os fabricantes não parecem interessados em disponibilizar estas atualizações para produtos antigos, já que o próprio movimento de adoção do IPv6 está criando mercado e gerando demanda por novos aparelhos compatíveis com o novo protocolo.

Um problema para a adoção imediata deste novo protocolo é custo adicional em termos de recursos de rede que ele demanda. Pois o cabeçalho definido para os pacotes nesse protocolo apresenta um acréscimo de tamanho considerável em relação ao padrão vigente. Mesmo sendo um protocolo com um cabeçalho reduzido, em termos do número de campos e atributos, em relação ao IPv4 ele é consideravelmente maior em sua representação binária, especialmente pelo tamanho dos campos de endereçamento que são quatro vezes maiores que seu precursor.Isto gera um impacto grande no custo de armazenamento e propagação de tabelas de roteamento e resolução de nomes de domínio.

2.3 Movimento de adoção

Em fevereiro de 1999, o IETF Deployment WG, órgão responsável pela coordenação e implantação dos padrões gerados pelo instituto, fundou o fórum IPv6 [28] para dirigir sua implantação ao redor do mundo. Até agora, já foram estabelecidos fóruns em mais de 45 países, além de diversas forças de trabalho. Em julho de 2004 o ICANN anunciou que os servidores de DNS raízes já haviam sido modificados para suportar ambos os protocolos, IPv4 e IPv6.

O IPv6 ainda representa um pequeno percentual dos endereços publicamente acessíveis [29], que ainda estão em grande parte dominados pelo IPv4. Porém, as iniciativas para implantação do novo protocolo já estão surgindo em diversos países.

O governo dos Estados Unidos estabeleceu um mandado [30] obrigando todos os fornecedores, civis e militares, a terem seus produtos adequados ao novo protocolo até meados de 2008. Essa iniciativa certamente irá apressar este movimento.

Os novos contratos realizados junto ao governo americano já se equiparam em valores, cerca de US$ 150 bilhões, aos contratos realizados à época do *bug* do milênio. Espera-se que os valores para a conclusão deste movimento superem em muito estas cifras.

Inclusive, para iniciar esta adequação, o governo americano já adquiriu também um bloco de endereçamento do tipo /16 que representa 281 trilhões de endereços.

Atualmente os Estados Unidos possuem quase um terço do total de endereços disponíveis pelo IPv4 [31]. Em contrapartida, a China possui mais usuários de banda larga do que o número de endereços alocados para o país.

Inclusive, a China estabeleceu um plano de 5 anos para a implantação do IPv6 em sua infra-estrutura de redes. Este plano foi chamado de *China Next Generation Internet* (CNGI). O principal objetivo estratégico deste plano é estabelecer uma presença significativa no *Cyberespaço* através da adoção prematura do protocolo IPv6.

O plano é apresentar ao mundo sua nova infra-estrutura de redes já nos jogos olímpicos de Pequim em 2008. Tudo, desde de câmeras de segurança, táxis e câmeras filmando os eventos esportivos, estará conectado através do novo protocolo. Estes eventos serão transmitidos ao vivo através da internet enquanto os carros conectados poderão obter informações de tráfego em tempo real.

2.4 Mecanismos de transição

Observa-se que o movimento de migração para o novo protocolo já está ocorrendo e cada vez mais em um ritmo acelerado. Porém, algumas questões ainda apresentam desafios para este processo. Entre eles estão à adequação das redes privadas, a infra-estrutura de telecomunicações de países menos desenvolvidos e os dispositivos de controle de tráfego tais como roteadores [32]. Isto leva a um cenário híbrido onde o IPv4 e o IPv6 terão de coexistir e se intercomunicarem durante um bom tempo. Para isso, alguns mecanismos de transição foram desenvolvidos.

- **Dual stack** - Consiste da utilização de ambos os protocolos em um único nó. Atualmente, a grande maioria das implementações IPv6 utiliza este mecanismo. Ele permite que um nó se comunique com nós que utilizem em ambos os tipos de protocolo. Porém, exatamente pela característica híbrida ele não resolve a questão do endereçamento, pois o nó continua necessitando de endereçamento IPv4. Será utilizado principalmente nos servidores para

atender a clientes que suportem cada um dos protocolos. Seu principal objetivo é manter a compatibilidade. Essa técnica não permite que nós que utilizem apenas um dos protocolos se comuniquem com nós que utilizem o outro protocolo;

- **Tunelamento** - Consiste no encapsulamento de pacotes IPv6 dentro dos pacotes IPv4 tradicionais. Dessa forma, aquelas redes já adequadas ao protocolo IPv6 poderiam continuar a acessar as redes IPv4 sem consumir recursos de endereçamento IPv4. Isso pode ser feito diretamente em pacotes IPv4 através do protocolo 41 ou ainda através de pacotes UDP. O UDP seria utilizado para viabilizar a conectividade e fazer a travessia de NAT e firewalls. Neste caso, o protocolo ICE poderia ser utilizado para favorecer a conectividade e a qualidade do túnel criado. Dentre esses mecanismos existe o Teredo, que é um algoritmo de tunelamento automático via UDP que utiliza o STUN e o ICE para determinar a topologia de rede e encontrar o melhor caminho para o túnel criado;

- **Tradução** - Consiste da tradução de um pacote IPv4 em IPv6 e vice-versa. Um dos mecanismos que torna isso possível é através de um proxy em nível de aplicação, tal qual um web proxy. Esse tipo de solução requer configuração customizada em cada um dos clientes. Um mecanismo mais geral seria através de uma tradução que seja transparente para as aplicações clientes. Porém, isso requer que os dispositivos de rede, tal como NAT e firewalls fossem modificados para realizar tal operação. Alguns dos dispositivos existentes se mostraram ineficazes na prática e são considerados obsoletos. Existem pelo menos dois fabricantes, Datetek e Ambriel, que afirmam possuírem dispositivos eficazes de tradução que já possuem soluções para a maioria das questões encontradas anteriormente. Esses dispositivos seriam como um NAT entre IPv4 e IPv6;

Assim, pode-se observar que os problemas relacionados a exaustão do espaço de endereçamento estão gerando impactos sobre a conectividade dos nós conectados a Internet. De acordo com diversos estudos este problema ainda estará presente por algum tempo, logo soluções de conectividade e travessia de NAT tais como o ICE se fazem necessárias e prioritárias.

Capítulo 3

NAT - Network Address Translation

Neste capítulo trata-se do NAT, seus princípios básicos de funcionamento, suas classificações, seus impactos sobre a conectividade dos nós conectados e sobre os protocolos que passam pelo mesmo. É apresentado também o conceito de travessia de NAT e seus principais mecanismos com ênfase no UDP Hole Punch.

3.1 Conceito e Princípios Básicos

NAT [1] é um mecanismo de tradução de endereços de rede que têm como principal objetivo possibilitar a conectividade entre nós de uma rede privada com a internet. Assim, o NAT atua como um elo entre redes privadas e redes publicas.

O NAT não é um padrão formal administrado por alguma entidade, como ocorre geralmente com os protocolos de rede da internet. Mas sim, uma solução dada as necessidades originadas pelo a exaustão do espaço de endereçamento do protocolo IPv4. Ainda assim, o IETF criou o RFC 2663 documentando as principais considerações e terminologia utilizadas em sua implementação documento depois atualizado no RFC 3022.

Ele se popularizou especialmente pela forma como as redes privadas são endereçadas no protocolo IPv4. Ele lida com a incapacidade dessas redes serem roteadas nas redes publicas. Esse mecanismo se tornou bastante popular nos países que, por razões históricas, têm menos blocos de endereços alocados em média por pessoa.

Na prática, apenas os Estados Unidos possuem uma elevada quantidade de blocos de endereçamento *per capita* [31]. Ainda assim, os provedores de acesso a internet estimulam o uso de roteadores e modems equipados com NAT, o que faz com que esse país tenha também um dos piores índices de alocação racional desse espaço de endereçamento. Ocorre que existe um desperdício muito grande nos endereços disponíveis nos blocos alocados a este país.

O NAT se tornou um padrão em roteadores e modems domésticos (Ver figura 3.1) e para pequenos escritórios, o chamado mercado SOHO (*Small Ofice/Home Ofice*) . Assim, com a popularização do acesso a banda larga, esses dispositivos ganharam grande popularidade e difusão. O NAT também é amplamente utilizado por administradores de redes privadas, apesar de introduzir complexidade na comunicação entre nós e poder ter um impacto na performance da rede.

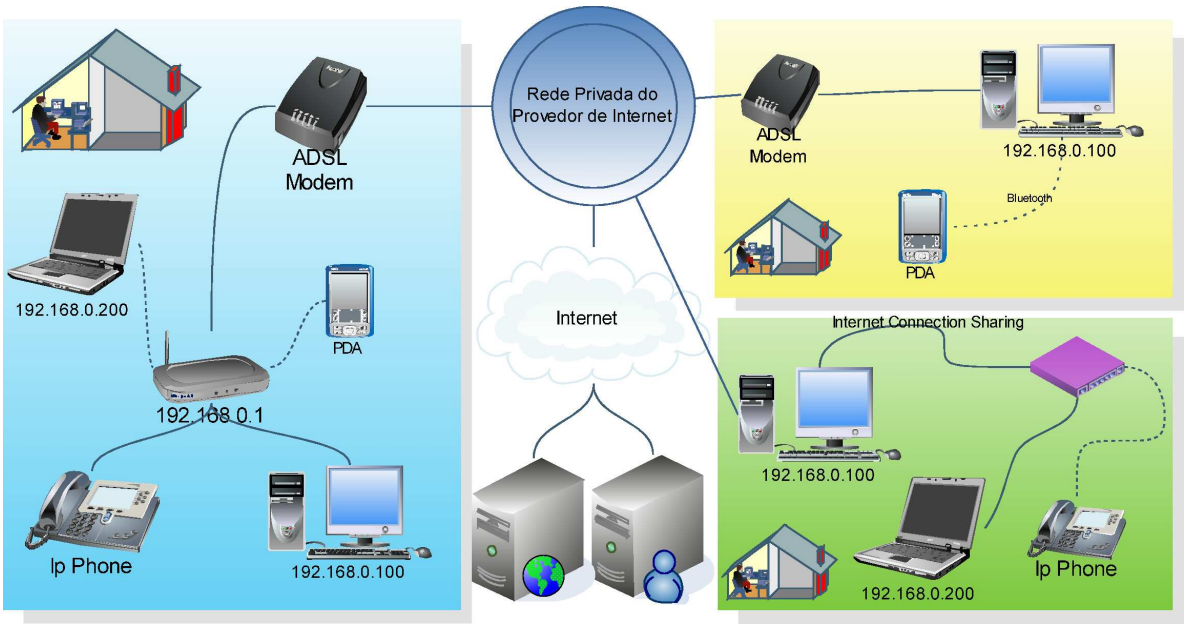


Figura 3.1: O NAT é amplamente utilizado no mercado doméstico e SOHO.

Seu princípio de operação consiste no compartilhamento de um endereço IPv4 publico e totalmente roteavel na internet pelos nós participantes de uma rede privada, os quais possuem endereços IPv4 reservados a essas redes privadas, e que, conforme exposto anteriormente, não podem ser diretamente roteados e unicamente identificados na internet. Assim, através do NAT, as redes privadas conseguem, de forma simples, imediata conectividade com a internet utilizando apenas um endereço IPv4, o que diminui muito a demanda por esse tipo de endereço.

Seu funcionamento se dá através de um roteador que atua como *gateway* e comuta todo o tráfego entre uma rede privada e uma rede publica. Ele reescreve os endereços de origem e/ou o destino de um pacote IP de forma a adequá-lo a rede de destino. Ou seja, ele converte um pacote que vem de uma rede para o formato necessário para ele trafegar e ser roteado adequadamente na outra rede. Esse mecanismo também envolve a reescrita de números de porta para os protocolos TCP e UDP de forma a viabilizar o mapeamento, em especial no sentido publico/privado. Os *checksums* também são reescritos de forma a validar as mudanças realizadas no pacote IP pelo NAT.

Em uma típica configuração, uma rede privada, também chamada de rede lo-

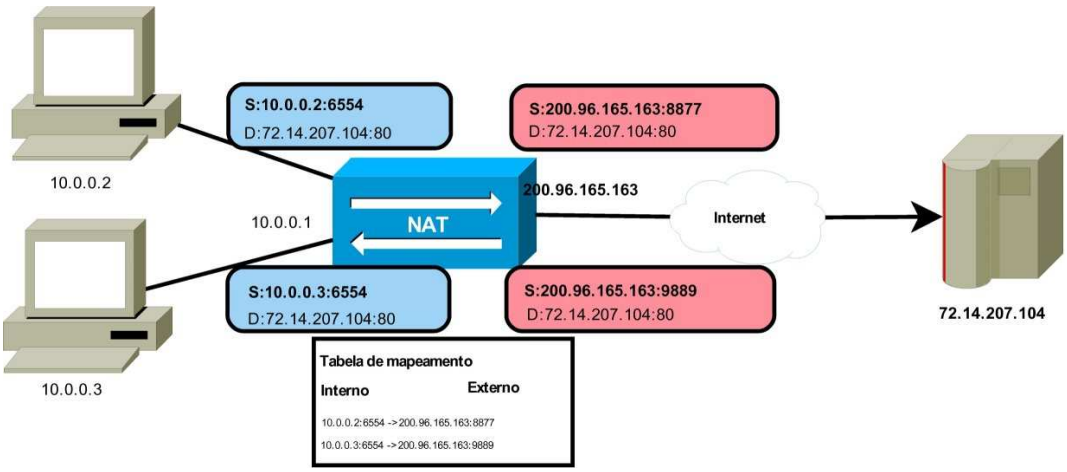


Figura 3.2: NAT realizando a tradução de endereços mantendo uma tabela de mapeamento.

cal, faz uso de endereços designados para uma das sub-redes previstas no padrão IPv4. Segundo o RFC 1918 [2] as faixas de endereços reservadas para redes privadas são 192.168.x.x, 10.x.x.x e 172.16.x.x até 172.31.x.x. Utilizando-se a notação CIDR eles são escritos como 192.168/16, 172.16/12 e 10/8. Como exemplo, suponha uma rede local que faz uso do espaço de endereçamento 10/8. Neste cenário (fig. 3.2), também existe um roteador nesta rede associado com um endereço privado que normalmente é 10.0.0.1 neste espaço de endereçamento. Apenas esse roteador está conectado a internet com um único endereço público, em geral atribuído e ele por um provedor de acesso a internet. Cada nó na rede privada é configurado para utilizar esse roteador como *gateway* de internet.

À medida que o tráfego flui da rede local para a publica esse roteador realiza a tradução do endereçamento em tempo real. Ele mantém os dados do pacote recebido alterando apenas os dados relativos ao endereço de origem para representar seu endereço público. Durante esse processo ele armazena os dados de estado de cada conexão entre nós da rede local e publica. Particularmente, os endereços e portas do nó local e do nó de destino são armazenados para cada conexão que se dá através deste processo. Quando um pacote é enviado no sentido contrário, da rede publica para a rede local, em geral em resposta a uma requisição realizada na rede local, ele utiliza estes dados para determinar para qual nó na rede local ele deverá encaminhar estes pacotes recebidos. Novamente será realizada a tradução, porém no sentido contrário, de forma a adequar o endereçamento de destino ao respectivo nó na rede local.

Para realizar o correto mapeamento de uma conexão entre um nó local e um nó na rede publica o roteador utiliza os endereços e portas utilizados na comunicação entre o roteador e o nó externo para realizar a demultiplexação entre as diversas conexões criadas e mapeadas. Assim, ao receber o retorno de uma requisição em determinada porta a partir de determinado endereço de origem o roteador é capaz de identificar qual nó interno deverá receber o pacote.

3.2 NAT e os protocolos TCP e UDP

Um NAT puro, que atua apenas no nível de rede do protocolo IP, pode ou não tratar corretamente protocolos que atuam exclusivamente neste nível, tal como o ICMP, dependendo de como o conteúdo do pacote é interpretado pelo nó externo ou interno. À medida que a aumenta-se a camada segundo o modelo OSI [18], a conectividade vai ficando cada vez mais prejudicada, a menos que o NAT execute algumas ações além da camada de rede. Isso pode ocorrer mesmo nos protocolos mais simples e imediatamente acima tais como o TCP ou UDP.

Isso ocorre pois os principais protocolos da camada de transporte, TCP e UDP, possuem checksums que validam todo o conteúdo enviado mais o cabeçalho específico do protocolo e ainda um "pseudo-cabeçalho"que contem os endereços IP de origem e destino. Têm ainda o fato que o conteúdo da camada de transporte pode ser fragmentado em vários pacotes IP ao passar pela camada de rede. Dessa forma, o NAT precisa tratar essas questões para viabilizar a comunicação através destes protocolos, do contrário os dados enviados seriam tratados como corrompidos ao chegarem aos seus destinos.

Assim, para um NAT permitir o correto funcionamento destes protocolos ele precisa recomputar também os respectivos *checksums* de acordo com a tradução realizada. Porém, devido à fragmentação que pode ocorrer na camada de rede, ele precisa armazenar todos os pacotes fragmentados para poder recalcular o *checksum* e então reescrever o cabeçalho corretamente no primeiro pacote IP a ser encaminhado. Este processo deve ocorrer nos dois sentidos da tradução.

Assim, fica claro que este é um processo que consome muitos recursos e que pode afetar bastante o desempenho de uma rede local que utilize o NAT para comunicação com a internet. Como otimização deste processo, alguns nós se comunicando fazem uso do algoritmo *MTU Path Discovery*, descrito no RFC 1191, para determinar o MTU no qual a comunicação na camada de rede pode ocorrer sem fragmentação. Assim, esses nós podem utilizar o bit "*don't fragment*"nos pacotes apropriados aliviando um pouco a carga no NAT envolvidos na comunicação.

Na prática ainda não existe uma solução geral e definitiva para estas questões. Esse é dos motivos pelo qual um dos objetivos da especificação IPv6 é evitar o uso do NAT por completo.

3.3 Aplicações afetadas pelo NAT

Alguns protocolos em nível de aplicação, tais como FTP e SIP, enviam os endereços IP junto ao seu conteúdo. Com a tradução ocorrida no NAT estes proto-

colos têm sua operação afetada, podendo ocorrer falhas.

Resolver essa questão, assim como na caso dos protocolos da camada de transporte, necessita-se entender o funcionamento destes protocolos para interpretar e modificar essas informações durante a tradução. Porém, neste caso, os protocolos da camada de aplicação se apresentam em universo de possibilidades e atualizações bem mais vasto, o que dificulta o tratamento do problema diretamente no NAT.

Uma das soluções empregadas para o problema é o uso de gateways em nível de aplicação. Em geral esse tipo de funcionalidade é encontrada em firewalls por ser um ponto central entre uma rede local e a internet. Ainda assim, para cada novo procotolo de aplicação é necessário um novo gateway específico, o que gera um grande custo de administração.

Outra solução possível é o uso de técnicas de travessia de NAT através de protocolos como o STUN e ICE. A travessia de NAT é possível em aplicações baseadas em TCP e UDP. Porém, o uso de UDP é mais simples, mais amplamente conhecido e mais compatível com dispositivos de NAT mais antigos. Em ambos os casos, o protocolo em nível de aplicação objeto, para tirar proveito desta abordagem, deve ter sido projetado levando a travessia de NAT em consideração. Ainda assim, essa técnica não é eficaz nos chamados NAT simétricos ou em outros tipos de NAT que não apresentam um comportamento adequado.

Outras possibilidades, não tão usuais por não serem diretamente compatíveis com a maioria dos dispositivos de NAT existentes, são o Universal Plug and Play (UPnP) e o NAT-PMP. Ambas as técnicas necessitam da cooperação do dispositivo de NAT, o que não ocorre na maioria dos casos.

Por outro lado, a maioria dos protocolos tradicionais em nível de aplicação do tipo cliente-servidor não necessitam de qualquer tratamento especial por parte do NAT. Atualmente, evitar complicações relacionadas ao NAT é um requisito prático no projeto destes protocolos em nível de aplicação.

NAT pode ainda dificultar uma comunicação na qual é empregada cifração através do protocolo IPsec. Pois em alguns casos, o endereço e/ou porta IP são incluídos na comunicação e por conta da criptografia não podem ser alterados pelos métodos descritos acima. Nesses casos, recomenda-se utilizar um protocolo de segurança que não inclua a porção dos dados relativa ao endereçamento, tal como o TLS. Ou ainda, se fazer o encapsulamento do canal IPsec através do UDP e utilizando o mesmo em algum método de travessia de NAT.

3.4 Tipos de NAT

Segundo a primeira especificação do protocolo STUN, RFC 3489, existem quatro principais classificações para NAT de acordo com as suas características de mapeamento e possibilidade de conectividade. Esta nomenclatura, ainda que não atenda exatamente a todos os tipos de NAT está se tornando cada vez mais utilizada para descrever o comportamento de um NAT com relação a conectividade.

Esta nomenclatura considera as características do mapeamento como observadas por nós externos. Segundo a primeira especificação do STUN são necessários pelo menos dois nós externos ao NAT para determinar com clareza em qual dessas categorias o NAT se encontra.

Ainda assim, onde couber será informado também a classificação usual utilizada pelos fabricantes. São elas:

3.4.1 NAT em cone total

Também conhecido como NAT um para um, pois, todas as portas de um endereço interno são mapeadas para um mesmo endereço externo específico preservando-se os números dessas portas. Ou seja, conhecendo-se o endereço publico do NAT e a porta associada a um nó interno, a comunicação pode ocorrer a partir de qualquer nó externo, não importando seu endereço ou porta.

Neste caso, conforme a figura 3.3, um nó externo pode enviar arbitrariamente pacotes para o nó interno bastando enviar os pacotes para o endereço e porta públicos mapeados no NAT.

Segundo a classificação mais tradicional de NAT este seria um NAT Estático.

3.4.2 NAT em cone restrito

São aqueles onde todas as requisições originadas de um mesmo endereço e porta internos são mapeados para o mesmo endereço e porta externos.

Neste caso, conforme a figura 3.4, um nó externo poderá enviar com sucesso um pacote para um nó interno apenas se o nó interno houver lhe enviado um pacote previamente, em qualquer porta. Este tipo representa a grande maioria dos dispositivos de NAT em uso, o que espera-se confirmar nesta pesquisa.

Segundo a classificação mais tradicional de NAT este seria um NAT Dinâmico.

3.4.3 NAT em cone restrito também por porta

Têm as mesmas características de mapeamento do NAT em cone restrito, porém, a restrição para a comunicação considera também a porta.

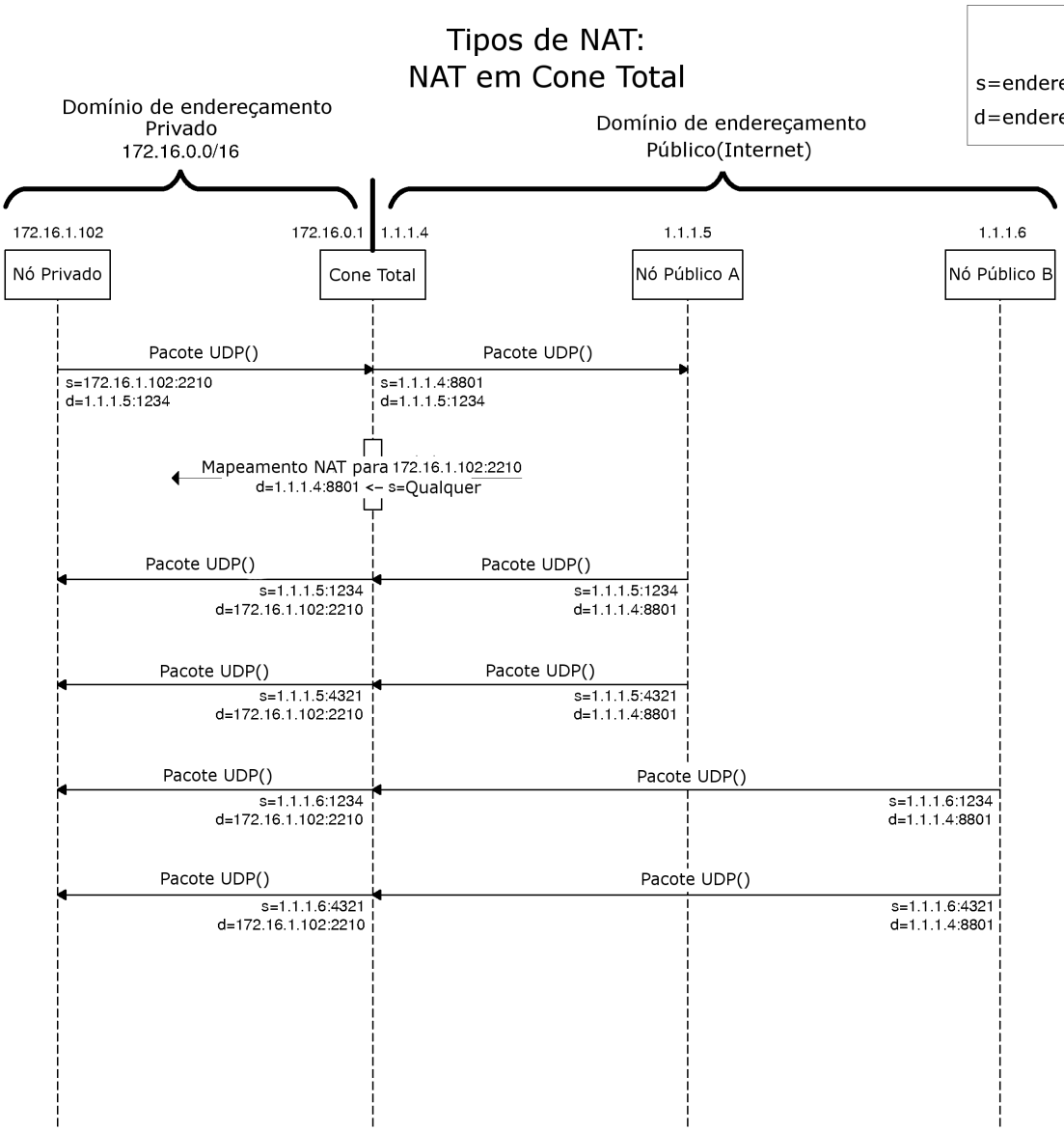


Figura 3.3: NAT em cone total.

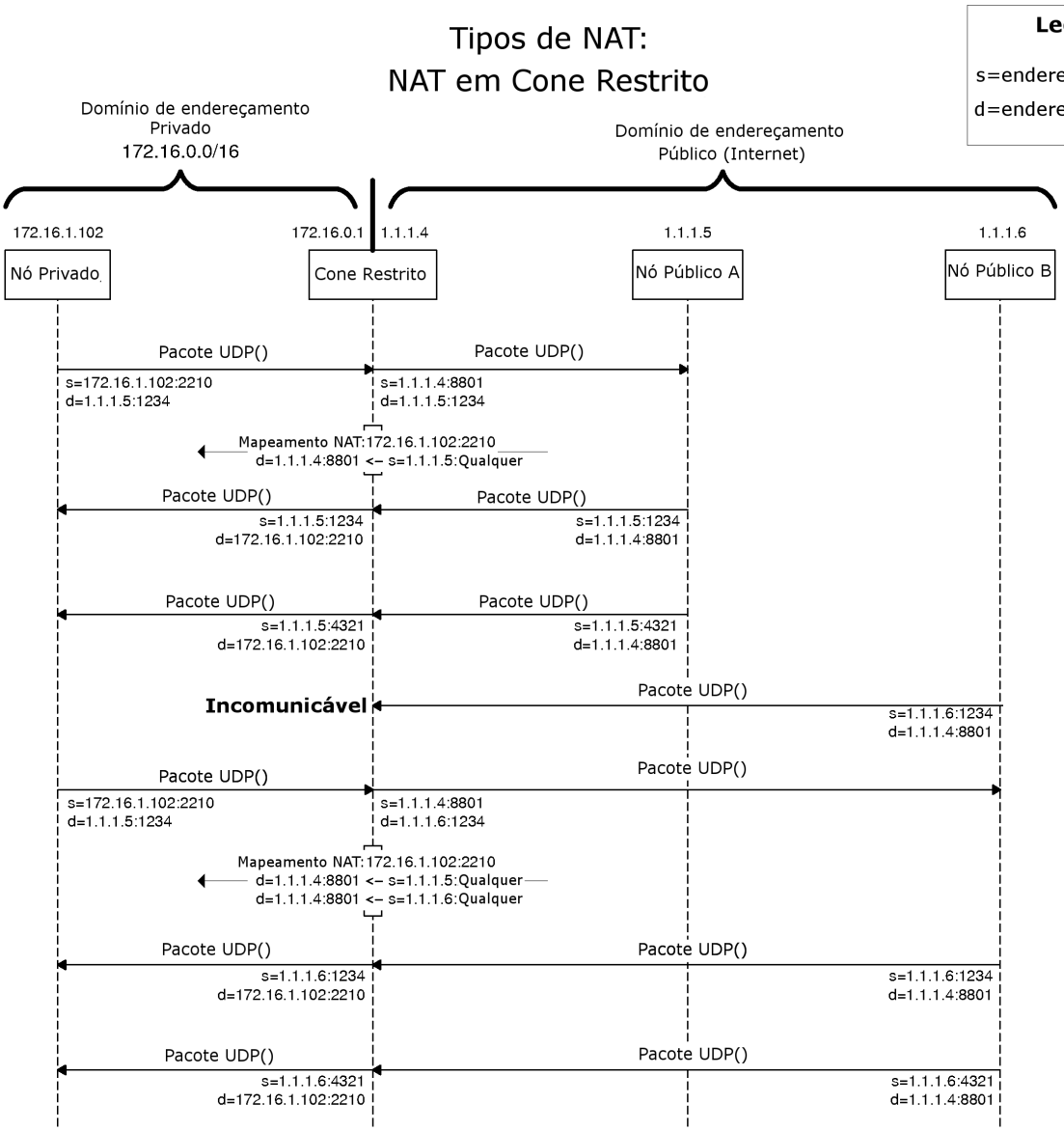


Figura 3.4: NAT em cone restrito.

Neste caso, conforme a figura 3.4.3, um nó externo poderá enviar com sucesso um pacote para um nó interno em determinada porta apenas se o nó interno houver lhe enviado um pacote através da mesma porta previamente.

Existem alguns NAT Dinâmicos que também são restritos por porta.

3.4.4 NAT simétrico

Cada requisição realizada a partir de um endereço e porta internos para um endereço e porta externos é mapeada unicamente no NAT. Ou seja, se o mesmo endereço interno envia uma requisição, com a mesma porta, porém para um endereço de destino diferente, um mapeamento diferente será criado no NAT.

Neste caso, conforme a figura 3.5, apenas o nó externo que receber um pacote do nó interno poderá responder. Isso inviabiliza que dois nós internos a dois NATs simétricos distintos possam estabelecer um canal de comunicação. Este tipo de NAT é utilizado com freqüência em redes coorporativas.

Segundo a classificação mais tradicional de NAT este seria um NAT Masquerade. Ele também é conhecido como *NAT Stateful* e é o atual padrão das implementações de NAT em gateways Linux através do IPTables. A idéia deste tipo de NAT é esconder, mascarar, a rede privada de rede externa. Exatamente este requisito de segurança que torna tão difícil o estabelecimento de conexões p2p efetivas neste tipo de NAT.

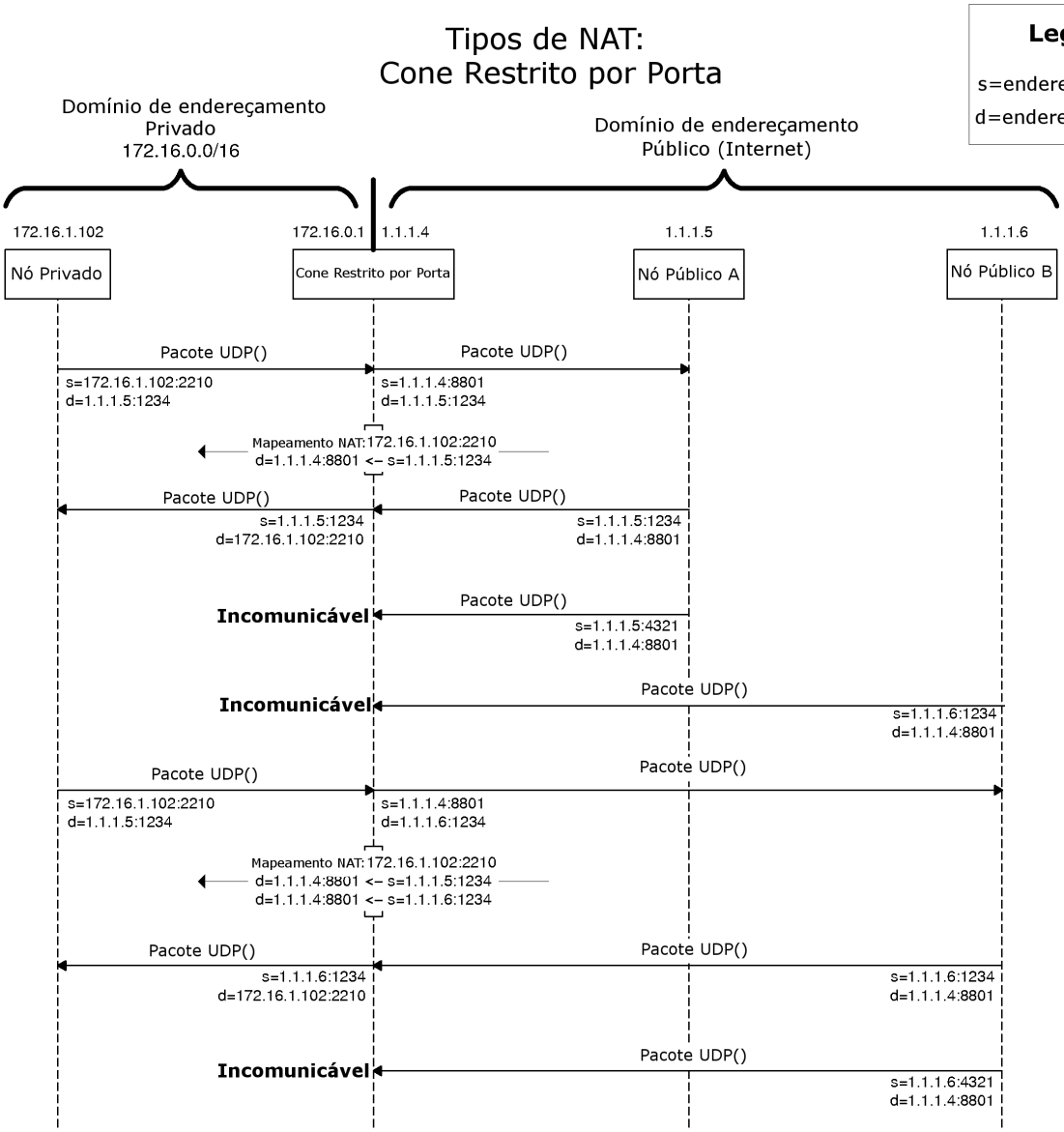
3.5 Travessia de NAT

A travessia de NAT refere-se aquele classe de algoritmos que tornam possível o estabelecimento de conexões entre dois nós que se posicionem entre um NAT.

A questão da travessia é muito abordada naquelas aplicações de comunicação cliente a cliente, especialmente em P2P e VoIP. Além disso, a comunicação via IPsec e algumas técnicas utilizadas na transição entre o IPv4 e IPv6 fazem uso da travessia de NAT.

Apesar de existirem diversas técnicas para se realizar a travessia de NAT, nenhuma é eficaz em todas as situações já que o próprio comportamento do NAT não é padronizado.

Em geral, a maioria dessas técnicas necessita de um servidor publico associado a um endereço IP publico e globalmente acessível. Algumas técnicas utilizam estes servidores apenas no estabelecimento de uma conexão, outras utilizam os seus recursos para o encaminhamento dos dados durante toda a comunicação.



NAT em cone restrito também por porta.

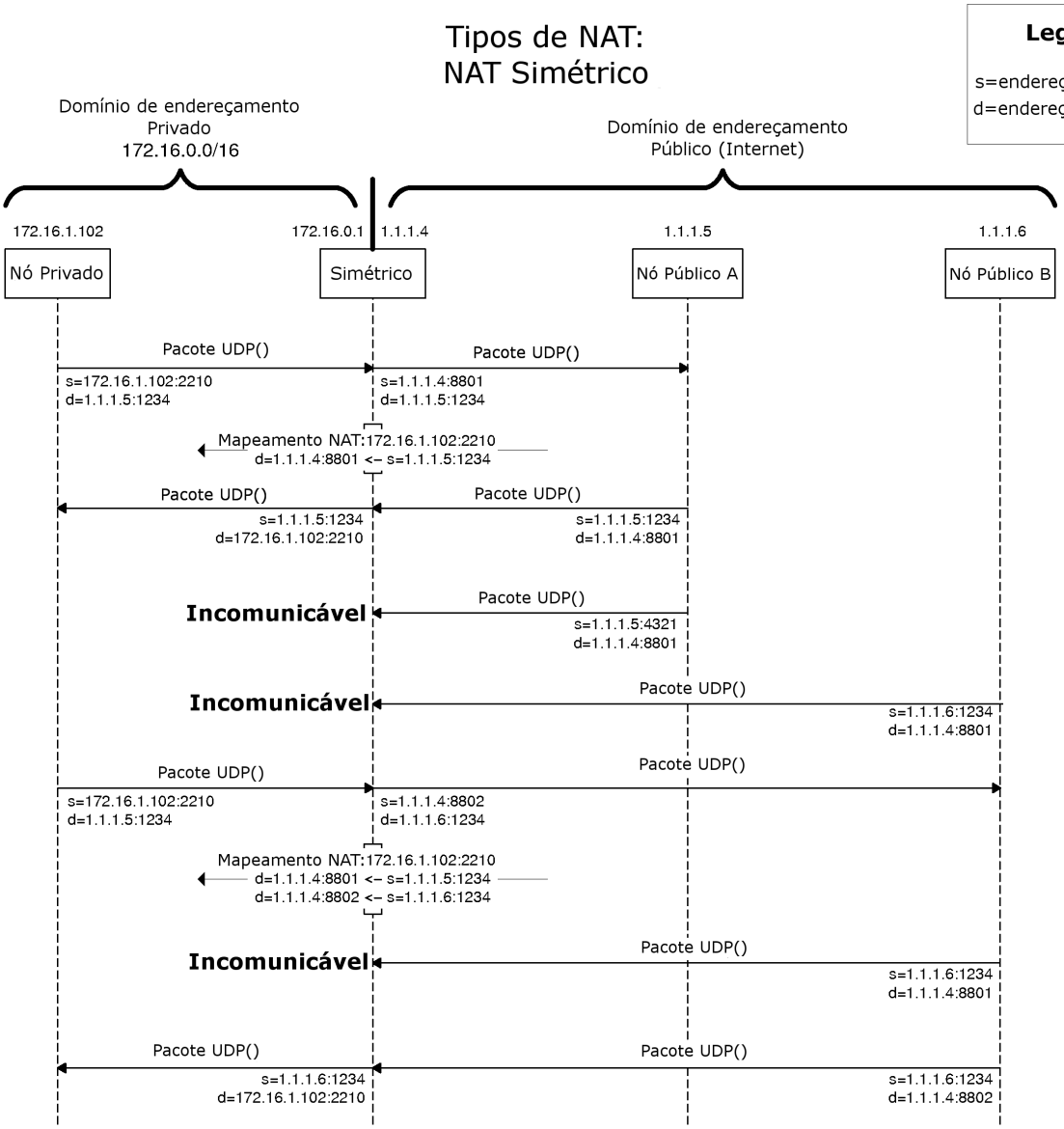


Figura 3.5: NAT simétrico.

3.5.1 UDP Hole Punch

O UDP Hole Punch [14] é a principal técnica usada para a travessia de NAT em nível de aplicação. Ele consiste de um processo coordenado entre ambos os nós que desejam se comunicar utilizando o protocolo UDP.

Segundo os comportamentos de NAT apresentados, em geral um nó não será capaz de receber pacotes de um outro nó externo se ele não houver enviado algum pacote para este nó externo previamente. Para contornar isso, o UDP Hole Punch apresenta um método coordenado onde ambos os nós enviam pacotes um para o outro simultaneamente. Fazendo-se isso, o NAT ao receber um pacote de um endereço para o qual seu nó interno já havia enviado um pacote, logo já havia um mapeamento neste NAT, ele irá encaminhar adequadamente este pa-

cote, permitindo assim o estabelecimento da conexão. Ou seja, ao transmitirem pacotes, um para o outro, simultaneamente os nós conseguem a travessia de NAT pois ambos configuram simultaneamente o respectivo mapeamento no NAT para viabilizar esta conexão.

Analisando-se a figura 3.6 pode-se perceber, que por conta do NAT, os nós *Cliente A* e *Cliente B* não conseguem estabelecer uma conexão direta seguindo o caminho *(b)*. Então dá-se início a um método coordenado de forma a viabilizar a travessia de NAT. Neste cenário, a conexão dos clientes com o servidor coordenador já foi estabelecida previamente, e logo ele já possui os dados destes clientes de como eles são vistos pela rede pública.

Para iniciar o processo, o Cliente A envia uma requisição ao servidor coordenador notificando que deseja realizar uma conexão com B *(1)*. Então, o servidor irá enviar, simultaneamente, os dados de conexão, dos seus respectivos pares, aos clientes que desejam se conectar *(2)*. Logo, o *Cliente A* receberá os dados do *Cliente B* e vice-versa. Ao receber estes dados, ambos os cliente iniciarão imediatamente o envio de dados *(3)* para o seu par seguindo o caminho informado pelo servidor *(a)*. Com isso, simultaneamente serão criados mapeamentos em ambos os NAT de forma a viabilizar o estabelecimento da conexão direta entre ambos os nós realizando a travessia de NAT.

Este procedimento é utilizado por diversos aplicativos de P2P e VoIP, em especial o Skype [13], porém suas implementações não são padronizadas e interoperáveis. Assim, um dos principais objetivos do ICE é criar um padrão de travessia utilizando-se deste conceito do UDP Hole Punch viabilizando o estabelecimento de uma infra-estrutura de servidores compartilhada e interoperável.

3.6 Conclusão

Assim, o NAT é um ferramenta muito útil para a ligação de redes privadas as redes públicas, em especial a Internet, que está se popularizando com velocidade, em especial no mercado SOHO. Nota-se que em geral, seu funcionamento ocorre de forma transparente e os usuários comuns ignoram que exista tal mecanismo. Ainda assim ele pode gerar impactos na conectividade destas redes privadas, em especial em aplicativos P2P e VoIP. Logo, é importante que existam soluções em nível de aplicação para realizar a travessia de NAT em conexões ponto a ponto entre redes privadas distintas.

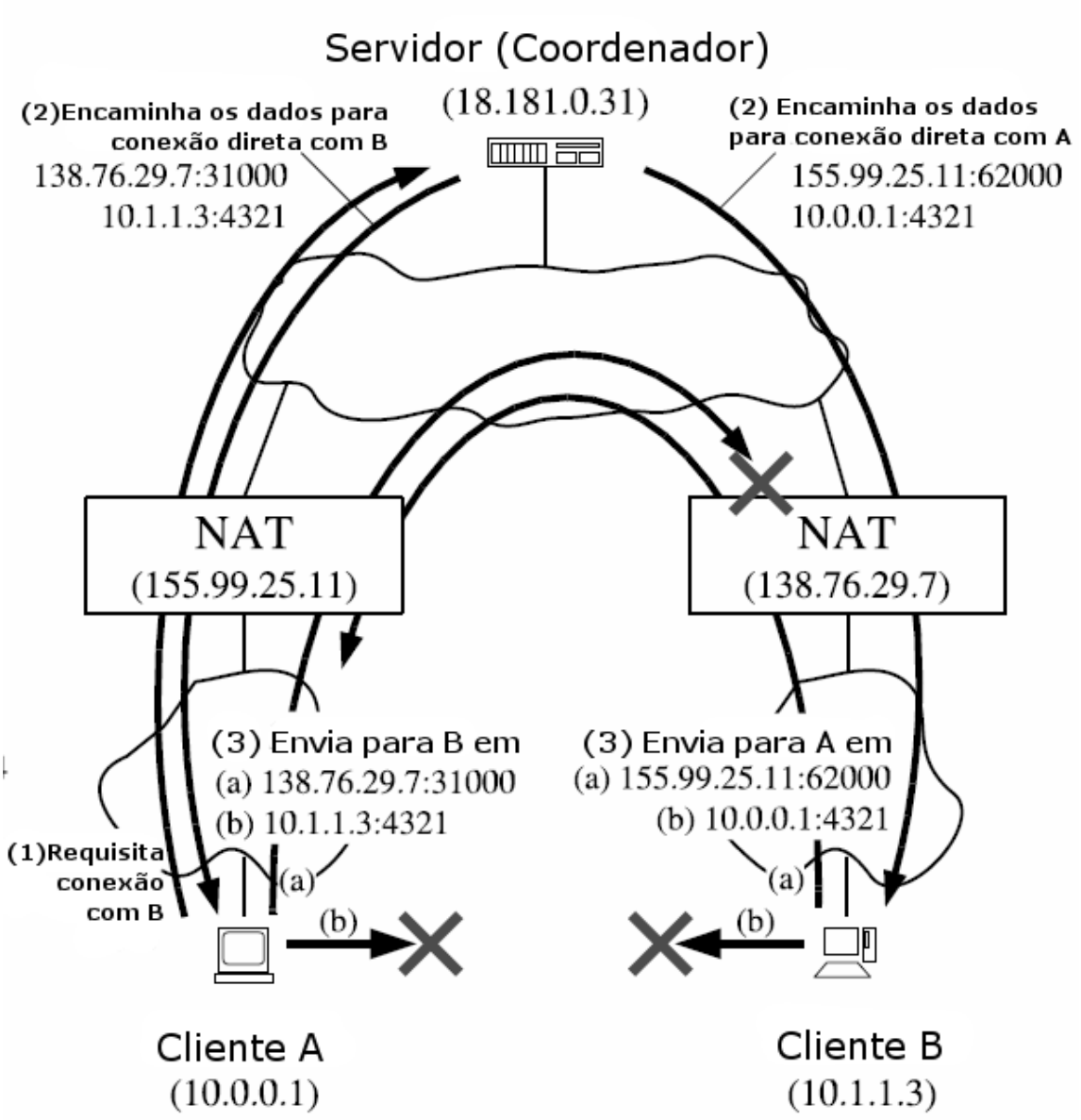


Figura 3.6: Funcionamento do UDP Hole Punch.

Capítulo 4

ICE - Interactive Connectivity Establishment

Este capítulo apresenta o protocolo ICE, os protocolos estendidos ou correlacionados e seu mecanismo de operação. Mostra que o ICE é um protocolo complexo, em especial na interação com diversos outros protocolos em nível de aplicação.

4.1 Apresentação

O ICE [12] é um protocolo para travessia de NAT para sessões multimídia estabelecidas sob o modelo Oferta e Resposta. Este protocolo surgiu da necessidade de se criar uma solução geral para a questão da travessia de NAT. Ele é essencialmente um protocolo em nível de aplicação cliente pois favorece a comunicação direta entre agentes que desejam se comunicar.

Isso porque, apesar de existirem diversas soluções, todas elas são ótimas apenas em determinadas aplicações e topologias de rede, mas apresentam diversas fragilidades em outras. O que ocorre é que os desenvolvedores e administradores estão tendo de fazer suposições a respeito das topologias onde seus sistemas serão implantados. Isso introduz complexidade e fragilidades ao sistema e certamente é indesejável.

O modelo de Oferta / Resposta (RFC 3264) apresenta algumas dificuldades em operar através de NAT. Seu principal objetivo é estabelecer um fluxo de pacotes de mídia direto entre os agentes o que permite se reduzir a latência, a perda de pacotes e ainda os custos de implantação das soluções. Para fazer isso, em geral se propaga os endereços e portas IP nas mensagens trocadas entre os agentes. Porém, como foi dito anteriormente, esse paradigma em geral não é tratado corretamente pelos dispositivos de NAT, o que impede uma boa conectividade.

O protocolo ICE é uma protocolo em nível de aplicação, logo utiliza-se dos recursos e/ou estende outros protocolos para viabilizar o estabelecimento de conexões multimídias entre agentes. Em especial ele estende os protocolos STUN e SDP com alguns métodos e atributos em suas mensagens. Além disso, ele

também compõe o próprio STUN e o TURN ao longo de seu ciclo de vida.

Ele também tem impacto direto nas implementações dos protocolos de fluxo de dados multimídia tais como o RTP. Isso porque neste caso o ICE têm de operar no mesmo endereço e portas associados a estes protocolos para viabilizar a travessia de NAT e neste caso deve ser feita a multiplexação dos pacotes entre ambos. O STUN oferece recursos para isso, porém a implementação do RTP têm de ser customizada para receber os dados apenas após o processamento do ICE. Na prática o ICE realiza uma composição pelo padrão *Decorator*.

Na prática, este protocolo congrega de forma padronizada um grupo de soluções e ferramentas para a travessia de NAT de forma a apresentar uma solução completa e robusta para o problema. O ICE é implementado como uma extensão aos protocolos STUN e SDP e faz uso direto do protocolo TURN (esses protocolos são abordados em mais detalhes abaixo). Ele ainda é considerado um *draft* no processo de maturação de protocolos do IETF, o que reforça a importância deste estudo e implementação para este processo.

4.2 Session Description Protocol - SDP

O protocolo SDP, documentado no RFC 4566, é um formato para descrição e iniciação de fluxos de comunicação multimídias. Este protocolo já se encontra em sua segunda versão desde julho de 2006, a primeira versão corresponde ao RFC 2327 de abril de 1998.

Seu principal propósito é exatamente o de descrever as características desse fluxo multimídia a ser estabelecido entre dois agentes, tais como tipos de mídia, formados de codificação, protocolos de comunicação e parâmetros de transporte tais como endereços e portas a serem utilizados no estabelecimento da comunicação.

O SDP não trata a respeito de como suas informações são transportadas, é agnóstico com relação a camada de transporte. Ele é puramente um formato para descrição de uma sessão multimídia para propósitos de anúncio e iniciação de sessões, ou seja é um protocolo de com parâmetros de configuração. Logo, em geral ele é encapsulado por diversos protocolos. No caso especial do modelo de Oferta e Resposta (RFC 3264) o protocolo geralmente utilizado para a iniciação e manutenção de sessões, e logo pela troca das mensagens SDP entre os agentes, é o SIP.

Suas mensagens são baseadas em texto, permitindo-se assim que estas sejam lidas e analisadas de forma simples. Porém seus atributos apresentam nomes bem resumidos em geral representados por um ou dois caracteres.

O SDP não implementa os fluxos multimídias de fato, ele apenas os descreve

para propósitos de anúncio e iniciação. Para o caso dos fluxos multimídias relacionados a comunicações de voz e vídeo em geral é utilizado o protocolo RTP.

4.3 Session Initiation Protocol - SIP

O SIP, cuja última versão refere-se ao RFC 3261, é um protocolo de sinalização amplamente utilizado para o estabelecimento e finalização de sessões de comunicações multimídia tais como chamadas de voz ou vídeo, *streaming* de audio e vídeo, aplicativos de mensagens instantâneas, jogos online, colaboração à distância, entre outros.

O SIP é um dos elementos da arquitetura IP Multimedia Subsystem (IMS) que visa implementar e entregar serviços multimídia para usuários de dispositivos móveis tais como celulares. O IMS é um dos padrões empregados pelo 3rd Generation Partnership Project (3GPP) que visa criar a infra-estrutura de terceira geração redes GSM.

O SIP é usado para se criar, modificar e terminar sessões de comunicação, entre dois os mais agentes envolvendo um ou mais fluxos de mídia. Os tipos de modificações envolvem modificações na camada de transporte, nos atributos dos fluxos multimídia, incluindo adição ou alteração desses fluxos e quais outras questões que precisem ser sincronizadas e acordadas entre os participantes de uma sessão.

Ele é um protocolo da camada de sessão segundo o modelo OSI ou da camada de aplicação segundo o modelo TCP/IP. Logo é um protocolo que não depende da camada de transporte. Em geral suas implementações utilizam como transporte o protocolo UDP e em alguns casos o protocolo TCP.

Ele permite aos agentes de uma rede se descobrirem e combinarem o formato para a sessão a ser estabelecida entre ambos. Com isso pode-se implementar, inclusive, o formato de estabelecimento de sessões utilizado no modelo de telefonia convencional (PSTN) envolvendo encontrar os agentes, solicitar estabelecimento de sessão, iniciar o chamado no aparelho solicitado e se for o caso estabelecer a conexão (Fig. 4.1). Cada um desses comportamentos pode ser acordado e definido através de troca de mensagens SIP durante o processo de início de uma sessão.

Durante o processo de estabelecimento de uma sessão o SIP utiliza-se de diversos protocolos para realizar a conexão propriamente. No caso do ICE ele atua como um dos principais meios de transporte para as mensagens SDP que determinam os fluxos multimídias a serem estabelecidos segundo o modelo de Oferta e Resposta.

Entretanto, o ICE não traz qualquer dependência relacionada ao protocolo

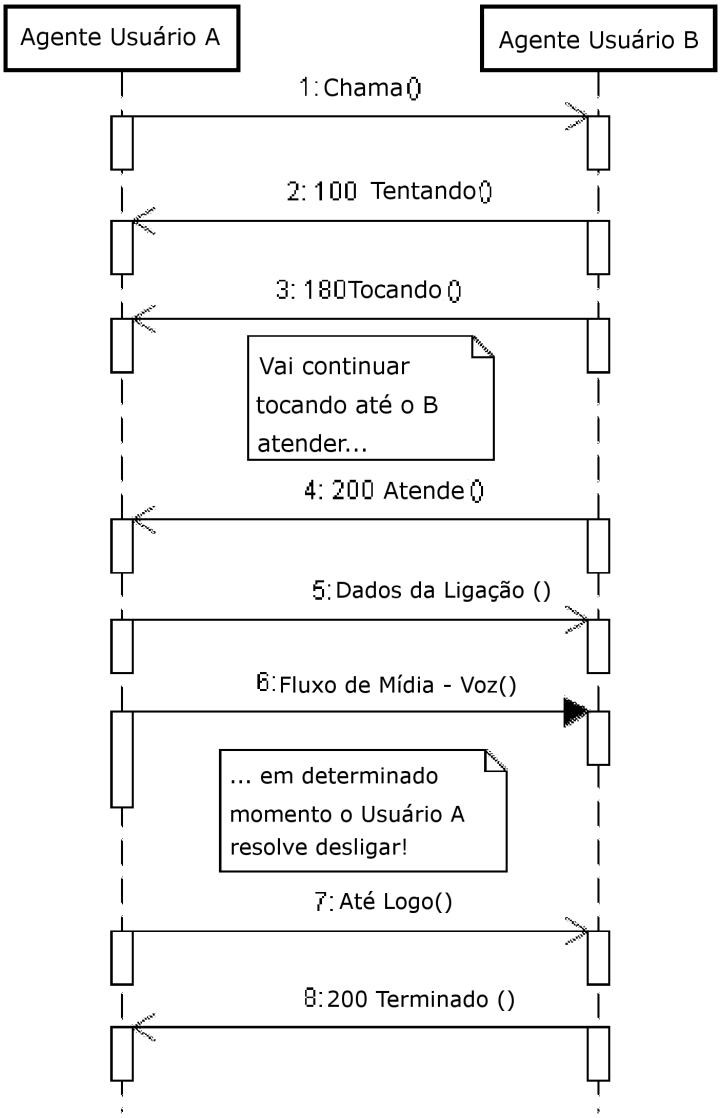


Figura 4.1: SIP usado em uma ligação convencional.

SIP, o ICE requer apenas que seja utilizado o modelo de Oferta e Resposta e que as mensagens que descrevem os fluxos multimídias seja encapsuladas segundo o SDP. O SIP é apenas a implementação padrão utilizada por este modelo.

As mensagens do protocolo SIP, assim como no SDP, também são representadas como texto simples, o que facilita sua análise e interpretação.

4.4 Real Time Transport Protocol - RTP

O RTP é define um formado padrão para pacotes de dados de audio e vídeo para serem transmitidos em tempo real via internet. Ele foi definido primeiramente pelo RFC 1889 em 1996 e foi substituído em sua versão mais recente pelo RFC 3550 em 2003.

Ele foi originalmente criado com um protocolo multicast, mas atualmente achou um grande nicho em aplicativos unicast. Ele têm sido freqüentemente usado em sistemas de streaming de mídia, videoconferência e em especial em tecnologias de VoIP.

O RTP foi desenvolvido em utilizando-se o UDP já que em geral as aplicações que o utilizam não são tão sensíveis a perdas de pacotes mas sim com relação a atrasos no envio e recebimento de dados. Dai inclusive decorre sua principal característica de ser um protocolo em tempo real.

As principais funcionalidades disponibilizadas pelo RTP são:

- **Identificação do tipo de payload** - Indica qual o tipo de conteúdo sendo transmitido por aquele pacote naquele momento;
- **Sequenciamento de pacotes** - Numeração e sequenciamento de pacotes para otimizar a reconstrução dos fluxos multimídias em possíveis desordenamentos ocorridos ao longo do roteamento de pacotes;
- **Marcação de timestamp** - Marcação de tempo nos pacotes para facilitar a sincronização e cálculo de parâmetros de QoS, em especial o jitter;

Ainda que o RTP procure ajudar ele não oferece recursos para garantir a entrega dos pacotes nem para fazer isso em tempo hábil. Também não oferece garantias sobre a qualidade de serviço dos dados sendo transmitidos. Estas questão têm de ser abordadas por outros mecanismos não previstos em sua especificação.

O RTP não define um conjunto de portas específico, o único padrão que ele segue neste sentido é o de alocar sempre portas pares para o protocolo UDP e a porta imediatamente subsequente para o seu protocolo de controle (RTCP). O RTCP é um protocolo de apoio ao RTP cuja principal função é a troca de informações relacionadas a manutenção do QoS do fluxo de dados.

Essa característica do RTP de não definir exatamente o intervalo de portas a ser utilizado o torna muito frágil com relação a questão da travessia de NAT e Firewall em geral. Isso porque não há como saber a priori quais portas serão usadas para quais clientes e logo não é possível fazer um encaminhamento de portas ou qualquer outra configuração neste dispositivos de forma a viabilizar a comunicação entre os agentes. A primeira versão do STUN e o ICE neste trabalho apresentados têm exatamente a função de resolver estes problemas para protocolos tais como o RTP.

4.5 Session Traversal Utilities for NAT - STUN

O STUN é um protocolo criado para a questão da travessia de NAT. Sua primeira versão foi definida no RFC 3489 onde seu propósito, chamado usualmente de STUN clássico, era um pouco diferente do atual: Simple Traversal of UDP through NATs. Esse protocolo tinha a função de identificar e caracterizar o tipo de NAT atribuído a determinado agente e realizar a sua travessia.

Ocorre que a abordagem apresentada por este primeiro documento se mostrou falha em alguns cenários e comportamentos de NAT e não se apresentava como uma solução geral o suficiente para a abordagem do problema, por exemplo não abordava o cenário onde a travessia simplesmente não era possível. Surgiu então sua segunda versão, sendo especificada como um *draft* no processo do IETF, chamado de RFC 3489 bis e intitulado Session Traversal Utilities for NAT. Este protocolo agora serve como base para outros protocolos, em especial o ICE, que juntos apresentam uma solução integral para a questão da travessia de NAT.

Ele, na prática e como o próprio nome diz, se apresenta como uma ferramenta a ser utilizada por outros protocolos ao lidar com esta questão. Seu funcionamento básico permite a um agente descobrir o endereço IP e porta externos mapeados ao seu endereço e porta internos por um NAT. Juntos esta tupla de 4 valores compõem o chamado endereço reflexivo segundo esta especificação.

Este protocolo, assim como seu antecessor, também pode ser usado de forma simples para se determinar o comportamento e características de um NAT, porém estas técnicas agora estão descritas em um outro documento que faz uso desta ferramenta, o também *draft* do IETF chamado NAT Behavior Discovery Using STUN.

O STUN é eficaz com diversos tipos existentes de NAT e não necessita de qualquer comportamento especial das mesmas. Porém, como foi dito, não é uma solução geral para a questão em si, ele se apresenta como uma ferramenta a ser usada neste contexto. Logo, define um formato de mensagem extensível que pode ser utilizado em diversos protocolos de transporte e ainda dispõe de duas formas de autenticação. Com isso, é possível implementar diversas soluções gerais de travessia de NAT tais como o TURN, o SIP OUTBOUND e o ICE.

4.5.1 Cabeçalho de uma mensagem STUN

Como um protocolo criado para servir como base para outros protocolos o STUN prevê alguns mecanismos em seu comportamento e na estrutura de suas mensagens de forma a viabilizar a extensão. Os principais recursos são as classes e métodos das mensagens assim como a utilização de uma codificação de atributos de tamanhos variáveis.

Como o protocolo STUN pode estar diretamente envolvido na transmissão de

fluxos de dados multimídia ele considera questões de performance e utilização de recursos com cuidado. Sendo assim, suas mensagens são codificadas em código binário usando um formato orientado a redes (bytes mais significativos primeiro, formato usualmente conhecido por *big-endian*).

O cabeçalho de uma mensagem STUN tem apenas 20 bytes e possui uma estrutura bem simples:

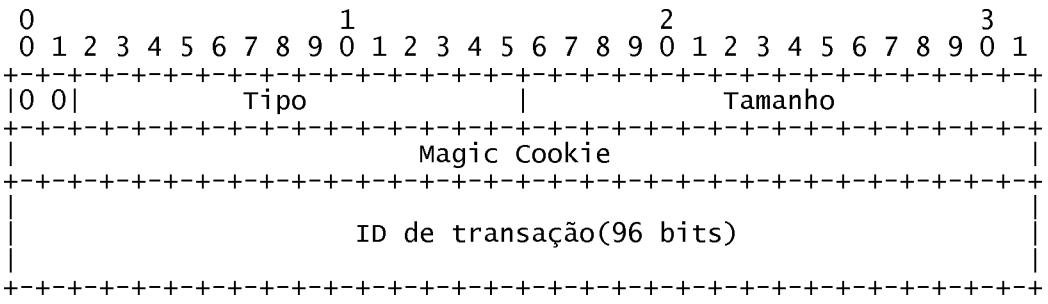


Figura 4.2: Cabeçalho de uma mensagem STUN (20 bytes).

Como podemos observar na figura 4.2 uma mensagem STUN está organizada da seguinte forma:

- **Bits de multiplexação** - Os dois bits mais significativos são sempre zeros e são usados para diferenciar de forma rápida pacotes de mensagens STUN em especial para os diversos cenários onde será multiplexado na mesma porta com outros protocolos;
- **Tipo** - Os próximos 14 bits representam o tipo da mensagem é uma composição da classe da mensagem com o método da mensagem (mais sobre a composição deste valor adiante);
- **Tamanho** - Com 16 bits, representa um valor inteiro contendo o valor do tamanho dessa mensagem em bytes desconsiderando os primeiros 20 bytes do cabeçalho. Ou seja, representa o tamanho em bytes dos atributos dessa mensagem, caso exista algum;;
- ***Magic Cookie*** - Contém o valor 0x2112A442 na ordem de rede, bytes mais signigicativos primeiro. Também usado para diferenciar uma mensagem STUN de outros protocolos;
- ***Id de transação*** - Identificador único de uma transação STUN com 96 bits. Usado para correlacionar requisições com suas respectivas respostas e para auxiliar na prevenção de alguns ataques.

O tipo de uma mensagem é a composição do valor da sua classe e o seu método. A classe de uma mensagem é um número de 2 bits e é usada para determinar o tipo de transação STUN sendo executada. As classes são requisição, resposta de sucesso, resposta de erro e indicação. Apesar de existirem 4 classes distintas

existem apenas dois tipos de transação suportados pelo STUN a transação Requisição/Resposta e a transação de indicação. As 3 primeiras classes se referem ao primeiro tipo de transação e a última ao segundo tipo.

A transação de Requisição/Resposta é o processo de um agente atuando como cliente STUN enviar uma requisição a um agente atuando como servidor STUN e este enviar uma resposta relativa a requisição recebida. Essa resposta pode ser positiva ou de erro, porém o cliente que originou a transação ficará sempre esperando por uma resposta neste tipo de transação, envolvendo inclusive retransmissões da requisição inicial no caso de retardo no recebimento da respectiva resposta.

A transação de indicação pode ser iniciada por qualquer agente STUN, tanto no papel de cliente ou servidor, e serve para informar a outra parte de alguma condição. Esta classe de transação não prevê retransmissões ou espera por respostas ou certificação de recebimento pela outra parte. A indicação é enviada e pronto, a transação é considerada finalizada.

As classes de mensagens são sempre as mesmas, mesmo em caso de protocolos que estendem o STUN com outras funcionalidades, vão sempre existir apenas estes dois tipos de transação. Para viabilizar este tipo de extensão o STUN oferece o recurso do método da mensagem.

O método da mensagem é um número de 12 bits e serve para identificar o uso sendo dado ao STUN naquele momento. O primeiro ponto de extensão do STUN é a definição do método sendo executado. Em conjunto com os métodos existem os atributos de cada mensagem. E cada método pode definir seu próprio conjunto de atributos, podendo alguns serem específicos para aquele método. O STUN propriamente define apenas o método BINDING que será detalhado mais adiante.

Sendo assim o tipo da mensagem pode ser obtido através de uma composição, na verdade um entrelaçamento dos bits, dos valores da classe e do método.

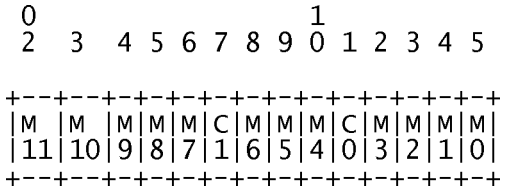


Figura 4.3: Composição do tipo de uma mensagem STUN de acordo com sua classe e método.

Observando a figura 4.3 podemos observar como são dispostos os 2 bits da classe e os 12 bits do método de forma a compor o número de 14 bits relativo ao tipo da mensagem. Este bits são dispostos segundo a ordem do mais significativo

(M11) até o menos (M0).

Como exemplo, a uma requisição do método binding tem a classe requisição (0b00) e o método binding (0b000000000001), logo seu tipo é codificado nos primeiros 16 bits da mensagem (os dois primeiros sendo obrigatoriamente zeros) como 0x0001. No caso de uma resposta do método binding temos a classe resposta com sucesso (0b10) e o mesmo método binding sendo o tipo codificado como 0x0101.

4.5.2 Atributos de uma mensagem STUN

Depois do cabeçalho de uma mensagem STUN podem haver zero ou mais atributos contendo dados necessários ao método STUN sendo executado assim como para validações gerais da mensagem.

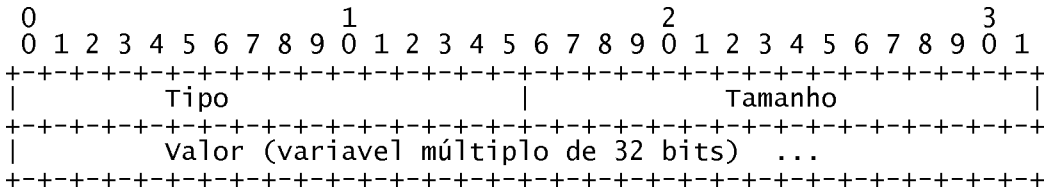


Figura 4.4: Atributo de uma mensagem STUN.

Cada atributo é codificado segundo o padrão *Type Length Value* (TLV) ou Tipo Tamanho e Valor. O tipo e o tamanho têm 16 bits cada um e o valor ocupa sempre um múltiplo de 32 bits. Caso o valor a ser preenchido não ocupe exatamente o espaço de um múltiplo de 32 bits seu valor será completado com zeros no final (bits menos significativos).

Qualquer atributo pode ser incluído mais de uma vez em uma mesma mensagem, e a menos que seja especificado de outra forma em alguma extensão, a ordem de inclusão tem relevância pois apenas a primeira ocorrência precisa ser processada pelo receptor, logo qualquer duplicação, a princípio será silenciosamente ignorada.

Os tipos de atributo caracterizam o mesmo, sua função e sua forma padrão de tratamento pelos receptores. Pensado-se em futuras revisões e extensões a este protocolo foram definidas duas classes gerais, dois intervalos de valores para estes tipos. Aqueles com valores entre 0x0000 e 0x7FFF são os atributos que necessitam compreensão pelo receptor. Ou seja, a implementação do protocolo STUN no receptor têm de conhecer os atributos desse tipo ao recebê-los, se isto não ocorrer será gerado um erro. Já para atributos com valores entre 0x8000 e 0xFFFF a compreensão pelo receptor é opcional e em geral são ignorados pelo mesmo.

O tamanho representa o número de bytes equivalente ao tamanho do campo valor deste atributo, antes dele ter sido completado com zeros para se tornar um múltiplo de 32 bits.

4.5.3 Comportamento básico

Considerando a estrutura básica das mensagens STUN e as duas classes de transações apresentadas o protocolo define os seguintes mecanismos para criação, envio e recebimento das mesmas para cada uma dessas classes de transação. Este comportamento poderá ser estendido em outros documentos para suportar novos métodos, atributos ou mensagens de erro.

A princípio qualquer agente atuando como cliente pode ter múltiplas transações concorrentes, com IDs de transações diferentes, com o mesmo servidor STUN inclusive. Porém, para evitar alguns ataques e abusos, um cliente deverá iniciar transações e realizar reenvio de mensagens seguindo uma política de *retransmission timeout* (RTO) e ainda se limitar a uma quantidade de 10 transações com cada servidor.

4.5.3.1 Tamanho máximo dos pacotes para envio de mensagens

Antes de realizar o envio deve-se verificar se o pacote da mensagem a ser enviada é menor que o MTU do caminho mais provável, se este for conhecido, isso evita que a mensagem STUN seja fragmentada ao longo de sua rota de transmissão. Se o MTU do caminho não for conhecido ele deve ser o menor entre 576 bytes e o MTU de primeiro salto para o IPv4 ou 1280 bytes para o IPv6.

Este valor equivale ao valor total de um pacote IP. Ou seja, para o IPv4, por exemplo, a mensagem STUN poderá ter no máximo 548 bytes (576 menos 20 bytes do cabeçalho IP, menos 8 bytes do cabeçalho UDP, assumindo que não se usou nenhuma opção do protocolo IP). Além da limitação de envio o STUN não trata o caso onde, apesar da requisição estar abaixo do MTU, a resposta seja maior que o mesmo e logo apresente fragmentação.

Na prática esta questão não foi vista como um problema para o protocolo STUN e suas extensões. Por outro lado, esta limitação é apenas um recomendação importante para o funcionamento padrão deste protocolo e não uma obrigatoriedade. Em casos onde o próprio STUN for usado para testar as características do MTU esta limitação poderá não ser aplicada.

4.5.3.2 Enviando uma mensagem STUN de requisição ou indicação

Considerando que as mensagens a serem transmitidas respeitam o formato de mensagem STUN definido anteriormente as transmissões das mesmas devem seguir

as seguintes regras.

Como este projeto prevê apenas a implementação do ICE e este define como protocolo de transporte o UDP para viabilizar a travessia de NAT por UDP Hole Punch trataremos apenas do envio de mensagens STUN por este protocolo de transporte. Porém, o STUN prevê sua utilização também por TCP e TLS-over-TCP além de permitir que outros protocolos sejam incorporados no futuro.

Ao realizar a transmissão de uma mensagem via UDP é possível que haja perda de pacotes. Neste caso, a garantia na entrega de pacotes em uma transação da classe Requisição/Resposta é realizada através de retransmissões realizadas pelo agente cliente que está realizando as requisições seguindo uma política de RTO. Porém, as transações envolvendo a classe indicação não utilizam este recurso de retransmissão e logo não possuem garantia de entrega.

Como sugestão visando a padronização desta política de RTO um agente cliente deverá retransmitir uma mensagem iniciando com um determinado intervalo para o RTO e ir dobrando o mesmo a cada retransmissão. O RTO deverá ser uma estimativa do *round-trip-time* segundo o RFC 2988 com alguns pequenos ajustes.

Primeiramente, o valor inicial para o RTO deverá ser configurável e deverá ser maior que 500 ms. Em segundo, a precisão para o cálculo do RTO deverá ser na casa de 1 ms e nunca arredondado para o segundo mais próximo. Por fim, não devem ser usadas as transações onde ocorram retransmissões para o cálculo do RTT.

As retransmissões devem continuar até que uma resposta seja recebida ou até que um determinado número de transmissões, que deve ser configurável, seja atingido. Como sugestão de padronização este número deverá ser 7. Após atingir este número o cliente ainda irá esperar por um período equivalente a 16 vezes o RTO. Se ainda assim não receber uma resposta válida o cliente deverá considerar esta transação como falha. Em caso de fortes erros ICMP (RFC 1122) a transação também será considerada falha.

4.5.3.3 Recebendo uma mensagem STUN

Ao receber uma mensagem STUN o agente, primeiramente, fará uma validação da estrutura desta mensagem se ela se aplica ao protocolo STUN e se sua estrutura é válida. Em linhas gerais, ele irá verificar se seus primeiros dois bits são zeros, se o *magic cookie* possui o valor correto, se o tamanho da mensagem é válido e existem dados no pacote correspondente ao mesmo e que o método declarado é conhecido e suportado por este agente.

No caso da mensagem ser de uma das duas classes de resposta o agente irá verificar se id da transação é válido e foi gerado por ele, ou seja, irá checar se esta é uma transação válida em curso.

Se o mecanismo de *fingerprint* para multiplexação avançada de pacotes STUN foi utilizado, logo existe um atributo com os dados de *fingerprint*, este será recalculado e checado. Este mecanismo ajuda a identificar pacotes STUN que estejam multiplicados com outros protocolos mesmo que estes possuam pacotes com cabeçalhos idênticos ao de uma mesagem STUN. Além disso, este mecanismo oferece uma forma de checar a consistência dos dados de uma mensagem. Ele consiste em incluir um atributo com os dados do cálculo CRC-32 de todos os outros dados que não ele mesmo. Este deverá ser o último atributo de toda mensagem STUN.

Por fim, o agente deverá verificar os atributos da mensagem e buscar aqueles que ele não compreende e possuem compreensão obrigatória e aqueles que ele compreende mas não eram esperados naquele momento. A partir deste momento o comportamento vai depender da classe da mensagem.

- **Requisição** - Se possuir mais de um atributo desconhecido que requer a compreensão do agente servidor, então este deverá retornar uma mensagem de resposta de erro e informar estes atributos.
- **Indicação** - Se possuir mais de um atributo desconhecido que requer a compreensão do agente receptor então será descartada silenciosamente e não deverá tomar qualquer ação com relação ao seu recebimento.
- **Resposta com sucesso** - Se possuir mais de um atributo desconhecido que requer a compreensão do agente receptor então será descartada silenciosamente e a transação será marcada como falha.
- **Resposta com erro** - Se possuir mais de um atributo desconhecido que requer a compreensão do agente receptor ou não possuir o atributo com o código do erro ocorrido então será descartada silenciosamente e a transação será marcada como falha.

4.5.4 Método BINDING

Como foi dito anteriormente, a primeira versão deste protocolo compreendia uma solução geral para a travessia de NAT. Esta porém, se mostrou falha e não tão geral como desejado. Então está sendo desenvolvida esta segunda versão com vistas a ser menos específica e sim como uma ferramenta ampla para tal. Sendo assim, ela define apenas um método chamado BINDING que visa ajudar os outros protocolos a abordarem de forma mais direta este problema.

O princípio de operação do método BINDING é simples. Ele se configura como um protocolo cliente/servidor. Quando um agente cliente STUN deseja saber o seu endereço reflexivo, isto é, o endereço e porta externos atribuídos pelo NAT ao seu endereço e porta privados na rede interna, este envia uma requisição utilizando o método BINDING para um agente servidor STUN que esteja externamente acessível. Ao receber esta requisição o servidor STUN verifica que

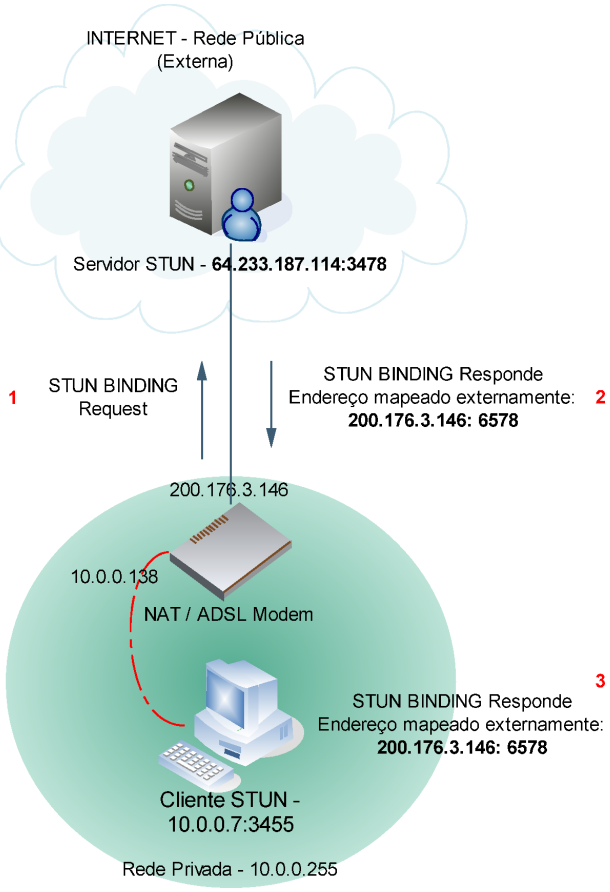


Figura 4.5: Cenário básico para utilização do método BINDING.

se trata do método BINDING e envia uma resposta contendo atributos com os dados do endereço e porta figurando na requisição recebida, ou seja os endereços do cliente como vistos por este servidor na rede externa. Endereços estes que foram mapeados e traduzidos pelo NAT no encaminhamento da requisição para este servidor. Assim, ao receber esta resposta do servidor o agente cliente poderá saber o seu endereço reflexivo para aquela transação.

Os dados do endereço reflexivo são enviados na resposta ao cliente através de dois atributos. O MAPPED-ADDRESS-ATTRIBUTE e o XOR-MAPPED-ADDRESS-ATTRIBUTE. Ambos os atributos guardam apenas os endereço e porta na rede externa associados a requisição participante da transação.

No primeiro atributo estes valores são armazenados sem considerar qualquer processamento ou mascaramento. Considerando que alguns NAT realizam o filtro e mapeamento de endereços inclusive no *payload* dos pacotes foi introduzido o segundo atributo que armazenada estes mesmos dados processados por um algoritmo de mascaramento através de uma simples função XOR. Isso faz com que o NAT não consiga alterar estes valores ao receber a resposta do servidor e logo eles possam chegar de forma íntegra ao cliente solicitante.

O processamento desta máscara XOR ocorre da seguinte forma. No caso da porta é realizado o XOR da mesma com os 16 bits mais significativos do *magic cookie*. No caso do endereço ser do IPv4 seu valor será o seu XOR com todos os bits do *magic cookie* e no caso do IPv6 será o XOR com *magic cookie* mais todos os 96 bits do id da transação.

Fazendo a análise do exemplo apresentado na figura 4.5 verifica-se que o cliente STUN está interno a rede privada 10.0.0.255, seu endereço nesta rede é 10.0.0.7. Logo, não é um endereço publicamente roteável na internet, o que sugere que este cliente esteja sendo servido por um NAT. De fato, de acordo com o exemplo, esta rede privada possui um modem adsl que realiza NAT para compartilhar seu único endereço externo com todos os agentes internos. Este modem possui endereços interno 10.0.0.138 e externo 200.176.3.146 e é configurado como o gateway de acesso a internet nos clientes internos através deste endereço interno.

Neste cenário, se este agente interno precisar saber qual o mapeamento realizado pelo NAT, relacionando determinado endereço e porta internos a um endereço e porta externos, ele pode utilizar os serviços de um servidor STUN acessível na rede externa. Para isso ele irá enviar uma Requisição STUN para o método BINDING para o servidor STUN, passo 1. O NAT ao processar esta mensagem irá trocar o endereço e porta internos do remetente pelos respectivos externos e guardar o mapeamento realizado. O servidor então irá gerar a resposta e incluir os atributos com os respectivos endereços e portas que figuram nesta requisição e enviar a resposta utilizando estes mesmos valores, passo 2. O NAT ao receber a resposta em seu endereço e porta externos irá acessar sua tabela de mapeamento e modificar o endereço do destinatário para os respectivos valores mapeados para o agente interno. Ao receber a mensagem o agente interno irá checar os atributos da resposta a sua requisição BINDING e com isso descobrir qual os endereço e porta externos estão associados ao seus internos, sabendo assim o seu endereço reflexivo, passo 3. Ele poderá usar este endereço reflexivo, por exemplo para realizar conexões ponto a ponto com agentes em outras redes externas a sua rede privada.

Enfim, o mecanismo de métodos no protocolo STUN na prática é o que define o que deve ser feito em cada agente envolvido na comunicação. Ele define quais os atributos devem estar presentes em cada mensagem bem como o ciclo de vida e máquina de estados envolvidos na execução de cada um destes métodos. Um protocolo que estenda o STUN em geral vai definir ao menos um método novo, com seus respectivos atributos. O ciclo de vida e máquinas estados definidos por este eventual protocolo pode ainda envolver mais de um desses novos métodos ou ainda utilizar os métodos existentes no protocolo estendido. Logo, um método é a única básica do funcionamento de cada protocolo que herda do STUN.

4.6 Traversal Using Relays around NAT - TURN

O TURN define uma extensão para se utilizar o servidor STUN como ponte de ligação entre dois agentes (*relay*). Esta extensão é útil para viabilizar a comunicação entre agentes que estejam posicionados entre um NAT e que por seu comportamento não conseguem estabelecer um canal direto de comunicação, tais como o NAT simétrico.

Como medida de segurança o TURN define mecanismos para evitar que o endereço e portas do servidor sejam utilizados para outros fins, tais como prover serviços tradicionais. Por outro lado, o TURN é uma alternativa que consome muitos recursos do provedor de serviços do servidor STUN, logo deve ser usado sempre como a última alternativa para viabilizar a comunicação entre dois agentes.

4.7 Visão geral do ICE

Seu funcionamento consiste em incluir uma multiplicidade de endereços IP e portas nas ofertas e respostas do SDP. Estes então são testados em relação à conectividade através de trocas de mensagens STUN entre os agentes. Alguns destes endereços candidatos incluídos no SDP também são identificados através do STUN e do TURN. Por conta desta multiplicidade de endereços e portas testados para cada sessão estabelecida, o ICE também permite a seleção otimizada de endereços em clientes multi-homed e dual-stack (LAN e WIFI ou IPv4 e IPv6, por exemplo) o que o garante ainda um papel essencial na migração do IPv4 para o IPv6.

No inicio do processo de estabelecimento de uma conexão, os agentes em geral ignoram suas próprias topologias. Em particular eles podem ou não estar atrás de um NAT (ou múltiplas camadas de NAT) porém eles não possuem, a princípio, tal conhecimento. Ainda assim, o ICE permite a estes agentes buscarem não apenas um caminho com conectividade entre os agentes, mas em muitos casos o caminho ótimo para esta comunicação.

As etapas do ICE consistem em:

- **Identificar endereços candidatos** - Antes de enviar e ao receber uma oferta SDP, identificar endereços candidatos para cada fluxo multimídia, em cada um dos agentes que desejam se comunicar. Deste processo de obtenção de candidatos participam os protocolos STUN e TURN;
- **Avaliação e priorização dos candidatos** - Os endereços dos candidatos encontrados são categorizados e priorizados buscando o caminho ótimo para a comunicação. Apesar da especificação propor um algoritmo de calculo desta prioridade ela também apresenta mecanismos que permitam a customização deste protocolo de acordo com o cenário. É imperativo porém

que ambos os agentes que desejam se comunicar utilizem o mesmo protocolo de priorização;

- **Envio dos endereços candidados** - Estes endereços serão incluídos tanto na oferta quando na resposta trocadas entre os agentes;
- **Checagem de conectividade** - Ao receber os endereços candidatos cada agente faz o emparelhamento dos candidatos locais e remotos e iniciam-se os testes de conectividade, respeitando-se a ordem e precedência das categorias e prioridades. Estes testes de conectividade se dão através do protocolo STUN;
- **Estabelecimento da sessão** - Ao encontrar um ou mais candidatos com conectividade para um determinado fluxo multimídia o protocolo ICE termina enviando uma oferta SDP de atualização visando informar o caminho ótimo a todos os agentes envolvidos nesta comunicação, em especial roteadores e pontos de acesso que implementem a marcação e retenção de pacotes visando a qualidade de serviços, e então o envio de dados pode ser iniciado;
- **Manutenção da conexão** - Após estabelecer a conexão o ICE irá executar o processo de manutenção do fluxo de dados através de *Keep Alives* enviados em intervalos regulares. Este procedimento também é realizado utilizando-se recursos do protocolo STUN.

Na figura 4.6 são apresentadas as etapas necessárias para que um agente implemente com sucesso o protocolo ICE junto a pilha SIP / SDP. Este cenário consiste de dois clientes (A e B) que desejam realizar uma ligação (VoIP) utilizando-se de uma conversação SIP padrão conforme apresentada na figura 4.1. Este funcionamento remete ao mecanismo de UDP Hole Punch. Isto porque ambos consistem em processos coordenados entre dois agentes nos quais esta coordenação é realizada por um servidor comum a ambos os agentes e publicamente acesssível. Neste cenário, o servidor coordenador é o servidor SIP Proxy / Registrar.

Considerando um agente que implemente o ICE, antes mesmo de ele enviar a Oferta SIP, ele já realiza um primeiro passo do protocolo ICE, que é obter os endereços candidatos *(1)*. Os endereços candidatos representam todos aqueles endereços pelos quais este agente pode estar acessível. Em especial, ele representa os endereços das interfaces de rede locais *(a)* e ainda os endereços reflexivos que são os endereços mapeados por NAT e firewalls caso existam em sua rede. Para descobrir os endereços reflexivos este agente faz requisições STUN para o método BINDING para servidores STUN publicamente acessíveis *(b)*. É importante ressaltar que estes endereços utilizam a mesma porta que será utilizada para o fluxo de mídia. Logo, no caso da requisição STUN está será feita da porta onde espera-se estabelecer o fluxo multimídia, de forma a criar um mapeamento no NAT significativo para esta porta.

Após determinar os endereços candidatos o agente cliente A irá codificar estes estes dados junto aos atributos da mensagem SDP que descreve o fluxo de mídia

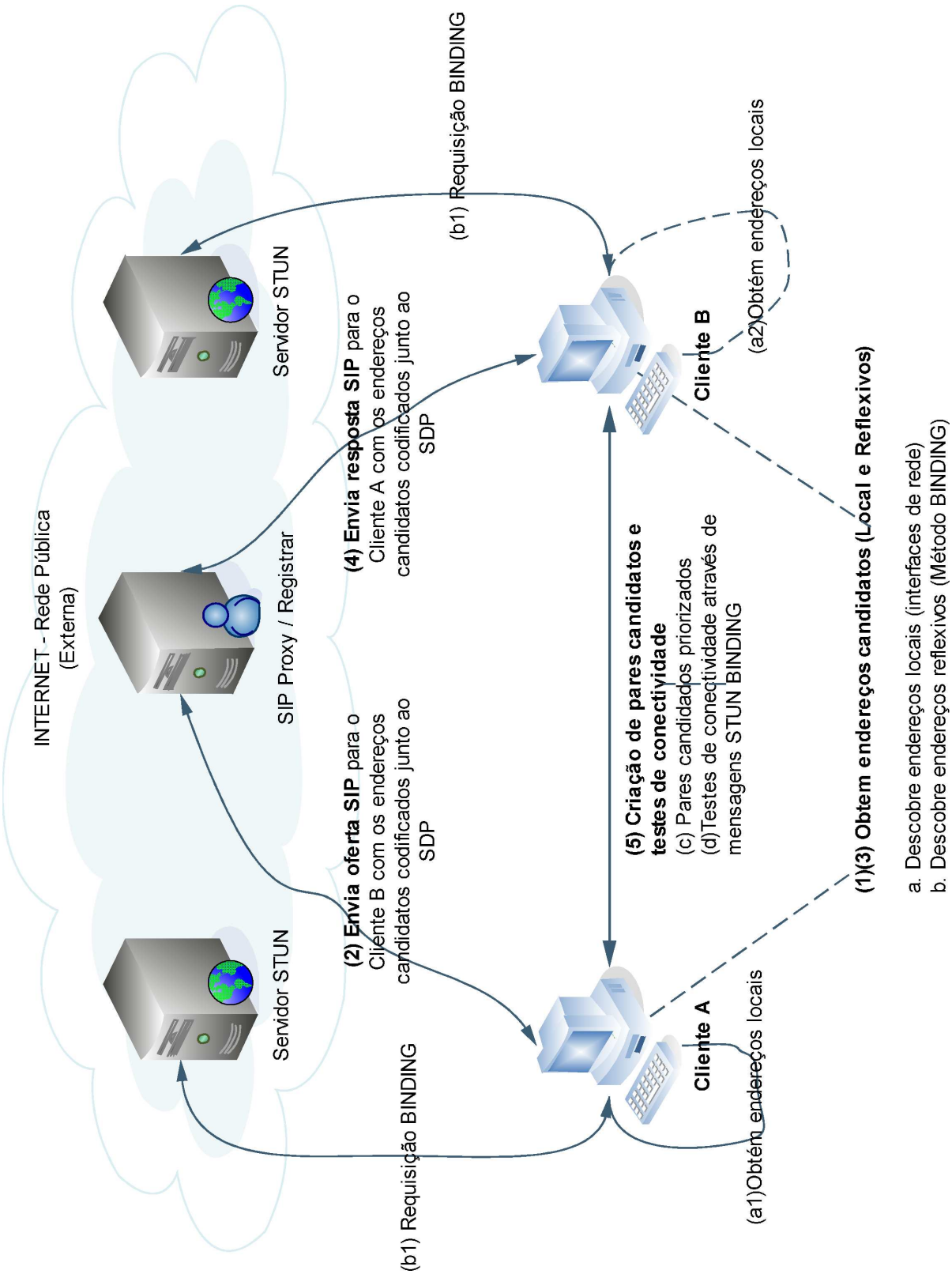


Figura 4.6: Funcionamento padrão do protocolo ICE.

que deverá ser criado e que está encapsulada na Oferta SIP a ser enviada *(2)*. Esta oferta SIP vai passar pelo SIP Proxy e chegar até o Cliente B.

Neste momento, o Cliente B toma ciência que uma nova Oferta foi realizada e que o Cliente Ofertante (A) está utilizando o protocolo ICE. Então, o agente cliente B também realiza o processo de obter endereços candidatos *(3)*. Estes endereços são codificados como atributos da mensagem SDP encapsulada na Resposta SIP a ser enviada para o Cliente A em resposta a sua oferta *(4)*.

Imediatamente após enviar a Resposta SIP o Cliente B dá inicio aos testes de conectividade *(5)*, pois este já possui os endereços candidatos do Cliente A que foram enviados junto a Oferta SIP. Já o Cliente A iniciará os mesmos testes assim que receber a Resposta SIP contendo os endereços candidatos do Cliente B.

Para realizar estes testes ambos os agentes criam uma lista de pares candidatos a serem testados. Esta lista consiste de uma combinação entre todos os endereços candidatos locais e remotos. Esta lista é então ordenada segundo a prioridade dada a cada par. A especificação deste protocolo sugere um algoritmo de priorização que favorece os endereços locais, porém as implementações podem adotar abordagens diferentes. São também sugeridas algumas heurísticas para melhorar a performance dos testes de conectividade, no sentido de não testar pares que possuam as mesmas características de pares já testados previamente.

Os testes consistem de requisições STUN para o método BINDING enviadas do endereço candidato local do par sendo avaliado para o endereço candidato remoto deste mesmo par. Assim com no caso do UDP Hole Punch, o algoritmo de priorização faz com que estas mensagens sejam enviadas simultaneamente em sentidos opostos. Isto faz com que, em cenários onde exista NAT com o apresentado, ocorra a travessia de NAT pois as mensagens STUN vão configurar o mapeamento necessário para o encaminhamento dos pacotes enviados pelo outro cliente.

Assim, com este processo, o protocolo ICE apresenta um método padrão e geral para o estabelecimento da conectividade entre dois agentes. Em especial favorecendo a travessia de NAT e a busca pelo melhor caminho de comunicação, inclusive entre clientes *multi-homed*, incluindo soluções *dual-stack*.

Capítulo 5

Implementação

Este capítulo aborda a implementação do protocolo ICE, principal objetivo deste projeto. Nele estão relacionados as decisões de projeto, as principais bibliotecas utilizadas para o desenvolvimento, em especial o Apache MINA, a estrutura de pacotes, as principais classes desenvolvidas e as dificuldades encontradas ao longo deste processo.

A biblioteca desenvolvida neste projeto recebeu o nome de JICE e será disponibilizada como código aberto segundo a licença Apache. Ela implementa os requisitos mandatórios da versão mais recente dos protocolos STUN e ICE. O protocolo TURN, por ser opcional no ICE, foi retirado do escopo deste trabalho, porém mantido no *roadmap* do projeto assim como os requisitos opcionais e eletivos destas especificações. Como as mesmas ainda estão em maturação no IETF, em vias de serem estabelecidas como RFC, este projeto continuará seu ciclo de evoluções junto a comunidade de código aberto em linha com esse processo.

Esta implementação foi realizada na plataforma Java. Ela é compatível apenas com a versão mais recente desta linguagem pois necessita de alguns recursos para o tratamento de agentes *multi-homed* e *dual stack*. A *framework* Apache MINA foi utilizada como principal ferramenta para a programação dos clientes e servidores STUN.

5.1 Decisões de projeto

As escolhas tomadas ao longo desta implementação foram validadas de acordo com as seguintes justificativas e motivações.

5.1.1 Plataforma Java

Como o ICE é uma extensão direta dos protocolos SDP e STUN e ainda faz a composição do protocolo TURN sua implementação envolve o processo de integrar e coordenar componentes que implementem estes protocolos. E ainda, ele é absolutamente integrado aos protocolos de fluxo de dados, tais como o RTP,

pois requer a utilização do mesmo canal de comunicação, endereço e porta, de forma multiplexada. Sendo assim é um requisito implícito que o ICE tenha de ser implementado especificamente para cada plataforma e a plataforma Java ainda não possui tal implementação.

Ainda que o STUN seja auto suficiente por si só, e logo existe uma interoperabilidade entre plataformas neste nível, o ICE não é e além de usar o STUN em sua forma primitiva, ele também o estende no papel de cliente e servidor para testes de conectividade. Neste cenário é possível que uma sessão do ICE faça uso de um servidor STUN em outra plataforma para obtenção de endereços candidatos e faça uso do seu servidor/cliente STUN para a checagem de conectividade. Logo, ainda que já existam algumas implementações do STUN e do ICE em outras plataformas estas não podem ser diretamente utilizadas na plataforma Java.

Por outro lado, há uma forte demanda por este serviço na plataforma Java pois nela existem diversos clientes de soluções que utilizam fluxos de dados no modelo de Oferta e Resposta, tais como a pilha VoIP/SIP, que precisam manter a capacidade de conectividade mesmo estando em redes que possuem NAT.

5.1.2 *Framework* Apache MINA

O MINA (Multi-purpose Infrastructure for Network Applications) permite o desenvolvimento rápido de aplicações de rede com alta performance e escalabilidade. Ela procura abstrair a complexidade de se programar clientes e servidores de rede. Com isso pode-se focar apenas em desenvolver o protocolo em questão e deixar os outros aspectos necessários a um bom servidor ou cliente por conta do MINA.

Ela faz isso através de uma API assíncrona orientada a eventos sobre vários protocolos da camada de transporte tais como UDP/IP e TCP/IP utilizando os recursos do Java New I/O. Essa API é a mesma, independente do protocolo de transporte utilizado. Além disso, é possível acoplar diversos filtros ao tráfego de dados de forma simples e definir as políticas de alocação de recursos e processos.

Nas versões do Java anteriores a 1.4, os desenvolvedores de aplicativos de comunicação tinham de utilizar um modelo de programação orientado a conexões blocantes, orientado a *streams*, utilizando-se das ferramentas do pacote *java.io*. Apesar da simplicidade deste modelo ele não oferece escalabilidade para aplicativos que precisem manter diversas conexões abertas ao mesmo tempo, como é o caso em servidores. Como as classes e métodos de acesso a rede fazem o bloqueio da aplicação enquanto aguardam por dados criou-se a necessidade de se iniciar novos processos ou *threads* para cada nova conexão. Este cenário consumia muitos recursos dos servidores, em especial de memória e processamento e não se beneficiava dos recursos de acesso a rede existentes nessas plataformas.

A partir de versão 1.4 foi introduzida o *New I/O*, um novo conjunto de ferramentas para viabilizar a programação em rede de alta performance. Agora com o pacote *java.nio* uma única *thread* é usada para tratar de todas as conexões abertas. Aplicativos que usem este recurso podem colocar a comunicação em um modo não blocante, o que significa que ele não precisa ficar mais esperando pela chegada dos dados. Agora, quando estes dados chegarem existem mecanismos que vão notificar o aplicativo deste evento. Além disso, foi introduzido o conceito de *buffer* para tratar o fluxo de dados. A vantagem deste método é que ele apresenta uma camada de abstração maior com relação a *streams* e, logo, não estão diretamente associados a chamadas do sistema para a leitura dos dados.

Por outro lado, este novo paradigma trouxe uma grande complexidade para a programação destes aplicativos. Em especial ficou bem mais difícil separar a lógica de negócios das atividades de comunicação. Para atender a esta lacuna foi criado o MINA. Seu principal objetivo é criar um interface simples que tire proveito deste novo modelo. Na prática o MINA faz bem mais que isso, pois ele aplica boas práticas ao utilizar estes recursos e fornece diversos serviços comuns a servidores e clientes que de outra forma teriam de ser programados para cada novo aplicativo. Enfim, o MINA se apresenta como uma abstração e uma padronização na programação de redes de alta performance em Java.

Alguns testes de performance preliminares [19] indicam que o MINA pode produzir aplicativos simples de serem desenvolvidos e de alta performance. Em alguns casos ele se equipara ou até supera seus pares desenvolvidos em linguagens mais tradicionais na programação em redes.

Como ponto negativo, no caso específico deste projeto, ao utilizar esta biblioteca fica mais complicado adequar os protocolos que se sobrepõem ao ICE no mesmo canal de comunicação em regime multiplexado. Neste caso, a camada de transporte para estes protocolos tem de ser refatorada para utilizar os recursos do MINA e muita vezes os recursos deste se sobrepõem aos implementados nestes protocolos.

5.2 Arquitetura do JICE

A arquitetura do JICE herdou algumas das características da arquitetura e do ciclo de vida do MINA. A primeira delas sendo a separação entre o código de acesso a rede da lógica de negócios em nível de aplicação. Ainda assim, o JICE procurou reduzir sua dependência ao MINA buscando racionalizar e conter seu uso ao máximo. Sendo relativamente simples, se necessário, incorporar outra biblioteca equivalente ou simplesmente remover o MINA.

Antes de apresentar a arquitetura do JICE, como ele se acopla ao MINA, seus possíveis pontos de extensão e como os protocolos STUN e ICE foram implementados deve-se primeiro compreender a arquitetura do Framework MINA.

5.2.1 Arquitetura do Framework MINA

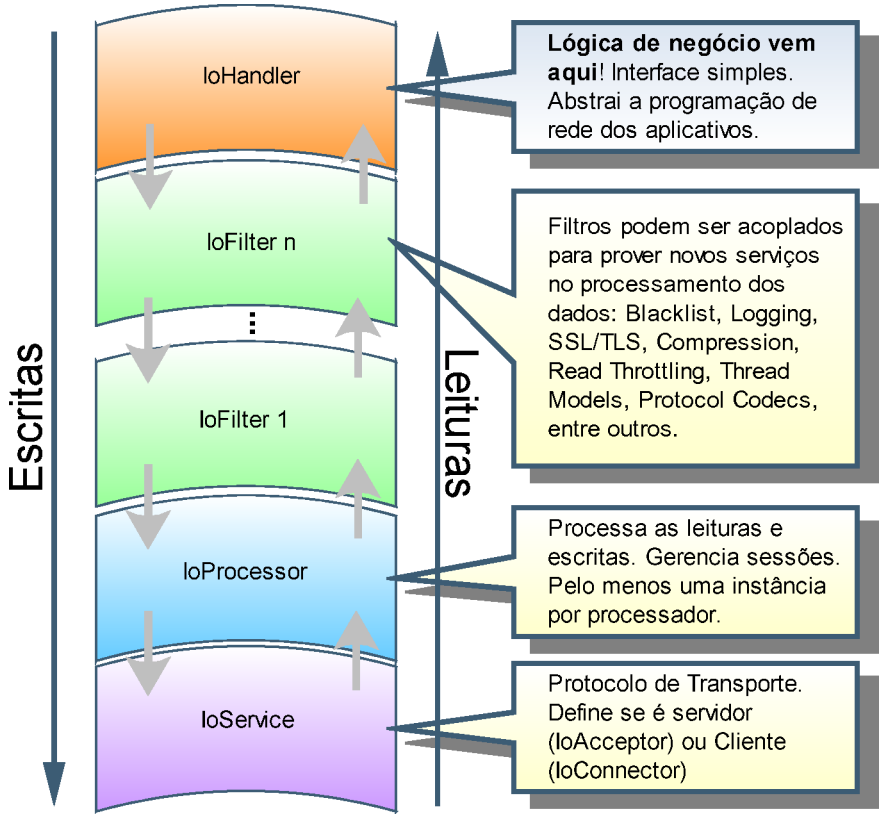


Figura 5.1: Ciclo de vida da Framework Mina.

A arquitetura do MINA [20] apresenta quatro níveis básicos que estão representados na figura 5.1. Estes níveis conceituais trazem uma abstração desde o menor nível da camada de transporte (IoService) até a camada superior de aplicação (IoHandler). As funções em cada nível, começando pelo nível mais baixo, são dispostas da seguinte forma:

- **IoService** - Este nível trata dos serviços de transporte de dados. Aqui são definidos o protocolo a ser usado (UDP, TCP ou In Memory), a alocação da interface, endereço, porta e o posicionamento (Cliente ou Servidor).

  Neste projeto foram utilizados o *NioDatagramConnector* como cliente e o *NioDatagramAcceptor* como servidor. Ambos permitem que se defina a interface de rede a ser usada assim como a porta de comunicação. Ambos utilizam o protocolo UDP e logo implementam o *IoService* para este protocolo.

- **IoProcessor** - É neste nível onde é feita a leitura e escrita efetiva dos buffers New I/O quando os dados são disponibilizados. Em geral, não é necessário configura-lo, o MINA trata desta questão automaticamente. Existe pelo

menos uma instância desse serviço para cada processador da máquina rodando o aplicativo. É neste nível onde as sessões são gerenciadas para cada conexão estabelecida, mesmo para protocolos não orientados a conexão tais como o UDP.

- **IoFilter** - Os filtros são um encadeamento de processadores de dados. Eles possuem as mais diversas funcionalidades, mas em geral eles atuam lendo e modificando os dados de uma comunicação de forma que o próximo filtro na cadeia já poderá enxergar estas mudanças. Os filtros são instanciados por conexão, são reutilizáveis e podem ser acoplados em tempo de execução.

  Existem diversos filtros disponíveis para aplicativos MINA. Existem filtros de codecs para conversão dos dados de buffer de bytes para formatos de mais alto nível chamados de mensagens. Por exemplo, formatos baseados em texto ou serializações para objetos Java. Pode-se inclusive criar filtros customizados que transformam certos objetos em bytes e vice-versa. Existem filtros que permitem realizar a diferenciação de pacotes para fins de multiplexação. Nestes filtros primeira é feita a identificação do pacote e depois o processamento deste é delegado para um codec específico. Este tipo de filtro foi utilizado no JICE.

  Existem também filtros para marcação e acompanhamento dos dados tais como Blacklist para banir invasores e endereços indesejáveis, Logging para acompanhar o funcionamento de determinado protocolo, SSL/TLS para acoplar o aspecto de segurança, compressão de dados, Read Throttling para implementar o controle da velocidade de leitura / recebimento de dados, ExecutorFilter para incorporar um pool de threads a cadeia de processamento de filtros, entre outros.

- **IoHandler** - O IoHandler é o nível de aplicação. É neste nível que a maioria dos aplicativos utilizando o MINA irá incluir seu código customizado relativo a sua lógica de negócios. Ele compreende uma interface simples com alguns métodos de callback que são chamados de acordo com o ciclo de vida de uma sessão de dados.

  Ele define métodos para o ciclo de vida da sessão propriamente dita, relatando seus diversos estados para o aplicativo. São eles: *sessionCreated, sessionOpened, sessionClosed* e *sessionIdle*.

  E métodos para o ciclo de vida de cada mensagem enviada ou recebida, são eles: *messageReceived* e *messageSent*

  Por fim, existe mais um método que é chamado em casos de exceção: *exceptionCaught*

  Apenas utilizando estes sete métodos uma aplicação pode incorporar sua lógica de negócios sem se preocupar como suas mensagens serão enviadas e como ocorrerão eventuais processamentos intermediários nas mesmas. Logo, a arquitetura do MINA favorece a separação de conceitos, a inversão de controle e a reutilização de código.

Sendo assim, um aplicativo padrão que faça uso do MINA deverá seguir a seguinte estrutura padrão como ponto de partida:

- Configurar um *IoService* para o protocolo e posicionamento desejados;
- Usar o *ExecutorFilter* como o primeiro da cadeia de filtros, a menos que se precise de uma latência realmente baixa, neste caso é melhor não usar este filtro;
- Usar um *ProtocolCodecFilter* para converter os bytes de rede em objetos Java de mais alto nível;
- Colocar a lógica de negócios dentro do *IoHandler*;
- Guardar estados e informações no *IoSession* que é informado em todos os métodos dos filtros e handlers;
- Utilizar a VM versão 1.5 em diante, pois somente a partir dela estão disponíveis os recursos do pacote *java.util.concurrent*.

5.2.2 Características gerais e decisões arquiteturais

Com o objetivo de obter uma solução eficiente e, ainda assim, minimizar a dependência ao MINA foram feitas algumas decisões arquiteturais importantes procurando balancear performance a acoplamento.

5.2.2.1 Objetos híbridos

A primeira decisão importante realizada foi como estruturar a codificação de uma mensagem STUN. O MINA apresenta dois modelos, um de baixo nível onde se trabalha diretamente com os bytes na forma como eles trafegam na rede, através de operações nos buffers disponibilizados; e um de alto nível onde estes bytes são primeiro mapeados para objetos Java antes de serem manipulados.

Do ponto de vista da estruturação do código e da simplicidade e estensibilidade da implementação o segundo modelo é mais interessante. Porém durante o processo de desenvolvimento, por questões de performance relacionadas a alguns requisitos especiais do protocolo STUN, optou-se por um modelo híbrido.

Este modelo híbrido consiste em objetos Java que operam buffers de dados disponibilizados pelo MINA e ao mesmo tempo oferecem uma interface de alto nível para acesso a estes dados. Especificamente foram criadas as classes *StunMessage* e *StunAttribute* neste paradigma. Estas classes guardam uma referencia ao buffer de bytes com os seus respectivos dados. Por outro lado, elas possuem métodos de alto nível seguindo o padrão JavaBeans para métodos de acesso [22](*geters* e *seters*) que permitem acessar e gravar estes dados nos buffers de forma simples.

Um dos motivos para usar estes objetos híbridos decorre da funcionalidade de fingerprint do STUN. O fingerprint deve ser calculado imediatamente antes do envio de uma mensagem STUN e novamente na validação de uma mensagem recebida. Ele requer que seja processado todos os bytes destas mensagem exceto os relativos a ele mesmo. No modelo puramente de alto nível isto significaria codificar a mensagem duas vezes, uma para processar o fingerprint e outra para realizar o efetivo envio. E mesmo no caso onde a primeira codificação fosse guardada ocorreria desperdício de recursos, pois os valores estariam guardados em duplicidade no buffer e no objeto.

5.2.2.2 Favorecer a extensão do STUN

Uma premissa muito importante considerada na elaboração desta biblioteca é que ela se trata de um ferramenta para facilitar a travessia de NAT. Como ela não é uma solução completa, pelas próprias características dos protocolos STUN e ICE, buscou-se criar uma *framework* que torne simples o uso, modificação e parametrização de suas funcionalidades. Desde o nível mais baixo do protocolo até suas camadas mais altas de ciclo de vida e máquinas de estado.

Por exemplo, com relação a capacidade e extensão do STUN, esta abordagem dos objetos híbridos é importante, pois permite a manipulação dos buffers diretamente nos objetos que implementam as mensagens STUN. De outra forma, seria necessário se estender também a implementação do *CodecFactory* criado para manipular as mensagens STUN. Atualmente este codec cuida apenas da multiplexação das mensagens e delega o seu real processamento para estes objetos híbridos e suas eventuais extensões. Para a simples manipulação dos dados nestes objetos híbridos, foram criados diversos métodos utilitários para auxiliar a leitura e gravação destas nos buffers da mensagem.

Técnicas de herança e composição foram amplamente utilizadas no projeto para este mesmo fim. Nesse sentido, quase todos os métodos e atributos foram marcados com o modificador *protected*. Isto permite que seus comportamentos e valores sejam facilmente alterados através de extensão. A composição, o uso de interfaces e do padrão Factory também permitem que os comportamentos possam ser alterados modificando-se o objeto que implementa alguns dos serviços da biblioteca. Com isso, espera-se que esta biblioteca possa acomodar as evoluções e inovações futuras do protocolo STUN e ICE.

5.2.3 Estrutura do pacotes e Diagrama de classes

Durante o processo de modelagem e considerando-se as características gerais e decisões arquiteturais adotadas a biblioteca JICE foi estruturada em pacotes e seus principais constituintes, suas associações, principais atributos e métodos foram mapeados em um diagrama de classes (fig. 5.2) para melhor entendimento e análise do projeto, seus pontos de extensão e evolução.

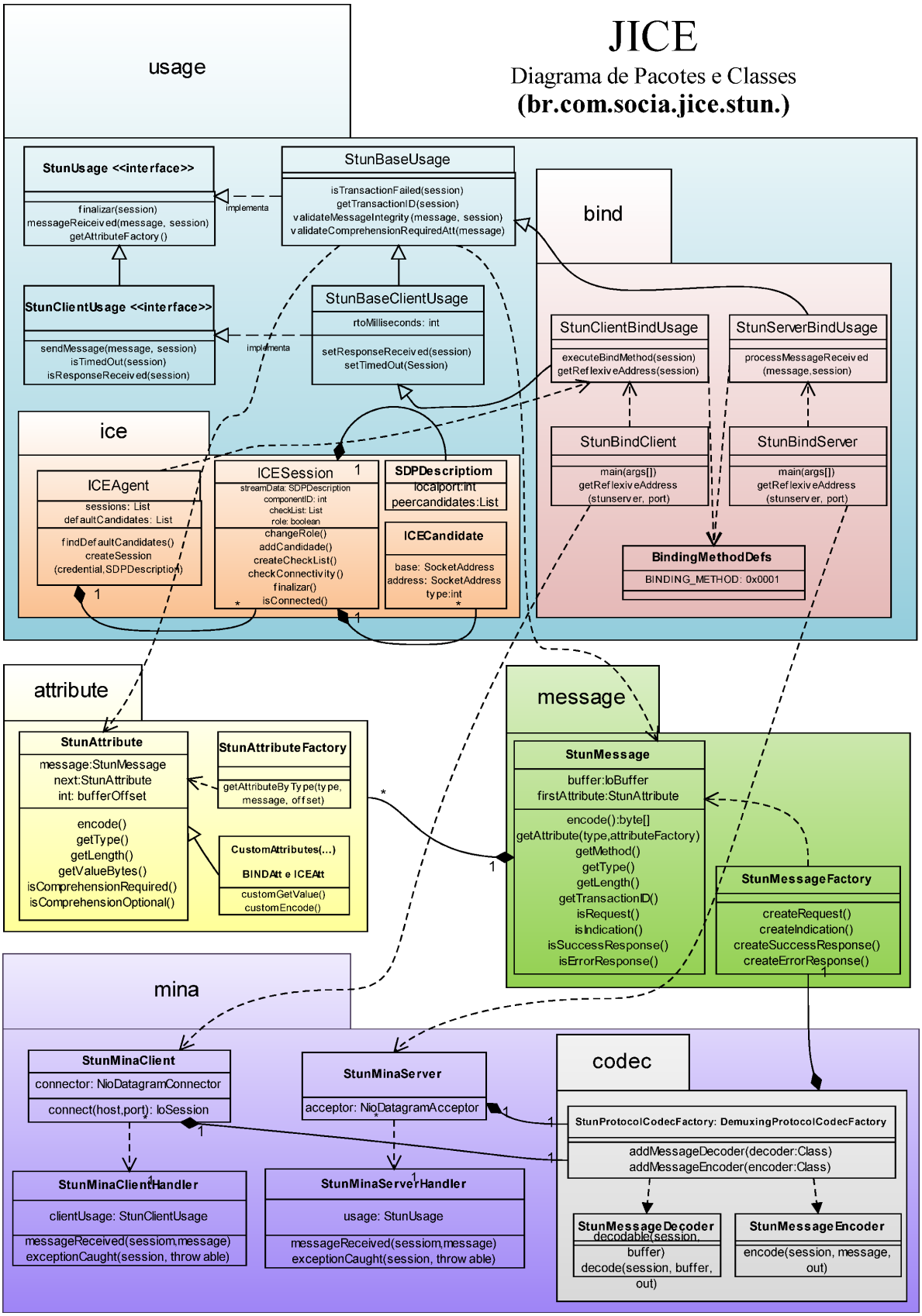


Figura 5.2: Diagrama de classes do JICE - Principais classes, métodos e associações.

5.2.3.1 Distribuição de pacotes e baixo acoplamento

A distribuição de pacotes teve como objetivo organizar o código e separar os objetos relacionados ao uso do MINA dos que implementam o protocolo STUN. Para isso, conforme representado na fig. 5.2, foram criados os seguintes pacotes:

- **br.com.socia.jice.stun.mina** - Aqui se encontram as classe de configuração e ligação do MINA com as demais classes. Seu objetivo é isolar conceitualmente o uso desta *framework* de forma a favorecer sua substituição caso haja necessidade.

  Especificamente, aqui estão as classes de cliente e servidor STUN que configuram o seu respectivo *IoService* usando um *NioDatagramConnector* ou *NioDatagramAcceptor* e também os dois *IoHandlers*, cliente e serivodr, que implementam a lógica de negócios STUN. Na prática estas duas últimas classes apenas delegam os eventos de mensagens recebida para um *Usage* do STUN.

- **br.com.socia.jice.stun.mina.codec** - Seguindo a decisão de isolar o MINA o máximo possível, neste pacote estão os codecs e *CodecFactory* responsáveis pela multiplexação dos pacotes MINA, delegação de pacotes para outros protocolos e leitura ou gravação de mensagens STUN propriamente ditas, através da utilização dos objetos híbridos.

  A classe *StunProtocolCodecFactory* estende a classe *DemuxingProtocolCodecFactory* e com isso viabiliza a simples utilização de outros protocolos junto ao JICE. Com ele é possível também se criar um stream de bytes para ser utilizado por outras aplicações que não usem o MINA.

- **br.com.socia.jice.stun.message** - Local do objeto híbrido que cuida da codificação e acesso de alto nível de uma mensagem STUN. Existe também uma classe utilitária que auxilia na criação e manipulação destas mensagens: *StunMessageFactory*.

- **br.com.socia.jice.stun.attribute** - Neste pacote estão as classes dos diversos atributos do protocolo STUN. Cada atributo definido no STUN, ou em suas extensões, utiliza ou estende o objeto híbrido *StunAttribute* para acessar e gravar seus respectivos dados no buffer.

- **br.com.socia.jice.stun.usage** - Neste pacote estão as classes que efetivamente implementam cada um dos usos STUN (chamado *Usage* ao longo da especificação). Um *Usage* determina os métodos, máquinas de estados, atributos e constantes necessários para implementar alguma funcionalidade específica dada ao STUN. O JICE implementa o *BindUsage* especificado no próprio STUN e o *ICEUsage* especificado pela extensão ICE.

  Um *Usage* na prática é uma classe que busca desacoplar e abstrair o *IoHandler* do MINA. Além disso, a classe *StunBaseUsage* implementa as principais regras de envio, recebimento e validação de mensagens STUN abrangendo retransmissões, verificações de transação e validação de fingerprint e logo é o ponto de partida para qualquer novo protocolo que faça uso ou estenda o STUN.

5.3 Dificuldades encontradas e suas soluções

Durante a implementação do JICE surgiram algumas questões, conceituais e técnicas, para as quais utilizou-se medidas corretivas ou ainda estas questões foram incorporadas a cultura do projeto.

5.3.1 Inexistência de tipos primitivos sem sinal em Java

Na linguagem Java não existem alguns tipos primitivos que existem em na linguagem C, por exemplo. Em especial, os tipos numéricos *unsigned* não possuem um correspondente nesta plataforma. Isto ocasionou alguns problemas durante o desenvolvimento do JICE em especial na fase de testes.

Por exemplo, durante a implementação do método Binding do STUN a verificação do número da porta do endereço reflexivo não estava de acordo com as leituras realizadas pelo Wireshark. Enquanto os pacotes mostravam determinado número de porta, o resultado produzido pelo Stun apresentava um número totalmente diferente, em alguns casos negativo, sem causa aparente para tal. Durante a depuração deste código verificou-se que as etapas de codificação e decodificação deste valor estavam corretas e ainda assim um resultado inesperado ocorria. Depois de algumas tentativas se chegou a solução e com isso entendeu-se o problema.

Ocorre que no protocolo STUN o número de porta é mapeado como um número de 32 bits. Durante a implementação utilizou-se um short (32 bits) para armazenar este número. Porém a codificação do STUN utiliza um valor *unsigned* e o Java não possui tal primitiva. Com isso, para armazenar este tipo de valor adequadamente é preciso aumentar a precisão da variável correspondente. Logo, no caso da porta a solução foi utilizar um int para armazenar este valor.

Este problema também serviu como justificativa para a utilização dos objetos híbridos. Isto porque no caso de haver um valor numérico com 128 bits ele teria de ser representado por algum primitivo que tenha mais de 128 bits e porém não há tal primitivo, o long é o maior primitivo numérico. Logo, estes valores têm de ser tratados como arrays de bytes mesmo e isto se torna simples quando se trabalha diretamente com os buffers pois eles possuem métodos utilitários para acesso e gravação destes dados.

5.3.2 O ICE não é simples de ser testado integralmente

Como foi dito, protocolo ICE, assim como o STUN, é uma ferramenta que estende o protocolo SDP e é usado por clientes que utilizem este protocolo no Modelo de Oferta e Resposta tal como o SIP. Logo, não é simples criar um ambiente que reproduza este cenário de forma a viabilizar a realização de testes integrais de sua funcionalidade.

Para abordar este problema foram utilizadas duas abordagens. A primeira sendo a de dividir e conquistar, ou seja, particionar suas funcionalidades em grupos e testá-los isoladamente. A própria estruturação do STUN favorece esta abordagem. Ela permite se testar isoladamente a estrutura das mensagens, o ciclo de vida dos usos, a eficiência dos clientes e servidores e corretude das máquinas de estado. Ainda que seja laborioso se testar e validar as funcionalidades segundo esta abordagem, esta se mostrou direta e muito eficaz.

Porém, para a validação e entendimento das necessidades de integração do JICE nos cenários para os quais ele foi concebido é necessário realizar testes de toda a sua funcionalidade integrada a estes cenários. Este caso sim, se mostrou um desafio e bem mais difícil de ser realizado. Uma das dificuldades foi integrar o MINA às ferramentas utilizadas para realizar este cenário. Esta integração foi realizada através de uma solução SIP simples e enxuta.

5.4 Solução SIP simples para testes

Conforme mencionado acima, para viabilizar testes de forma integral das funcionalidades do JICE foi necessário criar uma solução SIP que integrasse seus recursos.

Para isso, criou-se uma stack SIP completa e enxuta que reproduzisse um modelo de Oferta e Resposta e fizesse uso do JICE em regime multiplexado através de diversos cenários de NAT. Isso viabilizou os testes de conectividade e a pesquisa em cenários reais do JICE, agregando muito mais informações sobre os benefícios da sua abordagem.

Esta solução compreende a utilização da especificação jain-sip[23] e jain-sdp[24] como implementações da pilha SIP com atributos SDP em suas mensagens. Assim, foi criado um servidor sip muito simples para registrar e coordenar a criação das sessões pelo SIP entre agentes. Para integrar o JICE, foi criado um agente cliente que fizesse uso dos recursos do mesmo para estabelecer um canal de comunicação direto com outro agente cliente utilizando a mesma ferramenta.

Este agente cliente essencialmente é bem simples e foi implementado seguindo um dos exemplos de uso [25] destas bibliotecas. Ele é um cliente SIP básico que utiliza o SDP em suas mensagens para negociar um fluxo de dados *raw*, ou seja, sem codificação ou controle algum. Estes dados são textos transformados em bytes que foram alimentados por um interface que se assemelha a uma ferramenta de Instant Messaging bem simples.

Para viabilizar a travessia de NAT neste cliente foi utilizado o JICE em três níveis. No primeiro nível, ele foi configurado como um servidor STUN simples que implementa apenas o método Binding. No segundo nível ele foi usado para obter os endereços candidatos que foram então incluídos na oferta SDP, segundo as di-

retivas da especificação do ICE. E por fim, com base nos dados recebidos por esta oferta SDP em cada agente, ele foi utilizado para a checagem e manutenção da conectividade em regime multiplexado com o protocolo raw configurado pelo SDP.

Esta solução é extensível inclusive para cenários mais elaborados, onde seja utilizado um protocolo mais adequado para o fluxo de dados, tal como o RTP. Para isso, é necessário apenas utilizar alguma das implementações disponíveis e certificar-se que ela esteja integrada ao MINA, seja utilizando-se de seus recursos para o tráfego dos dados, seja utilizando alguma espécie de *Wrapper* da conexão para obter um stream dos dados já filtrados e multiplexados pelo JICE.

5.5 Conclusão

O JICE se mostrou um protocolo complexo e de difícil implementação. Parte pela sua correlação com outro protocolos, parte pelo grande número de otimizações e heurísticas previstos em sua especificação. Além disso, pode-se notar que a programação em rede na plataforma Java apresenta algumas questões o que sugere que esta questão ainda precisa ser trabalhada nesta plataforma. Prova disso é o próprio MINA, uma biblioteca de programação em rede de grande valia para este projeto.

Espera-se agora avaliar a solução desenvolvida com relação a performance e, principalmente, com relação a sua funcionalidade. Este trabalho está descrito no próximo capítulo.

Capítulo 6

Testes e Avaliação de Resultados

Este capítulo aborda os testes efetuados para avaliar a ferramenta desenvolvida, o JICE considerando sua compatibilidade e aderência funcional ao ICE. Busca-se também bases de comparação com outras plataformas em termos quantitativos e qualitativos. Nele estão descritos a metodologia utilizada, os cenários avaliados, os testes realizados e os resultados encontrados.

6.1 Metodologia

A metodologia de testes e avaliação de resultados para este projeto teve como principais objetivos determinar a eficiência e compatibilidade da solução desenvolvida frente a seu par existente em outra plataforma e determinar a eficácia da solução ICE em resolver a questão da travessia de NAT.

No primeiro caso, foi realizado um teste de carga do servidor STUN, em ambiente controlado, de forma a criar uma base de comparação entre as soluções de plataformas distintas (Java/MINA vs C). Neste caso, foi avaliada apenas a questão quantitativa do número de requisições atendidas realizando-se um comparativo entre as duas implementações (JICE e PJNATH).

Já no segundo caso foi criado um ambiente de testes que reproduza as principais situações encontradas pelos agentes reais que utilizarão o JICE. Em especial testar o protocolo com as diversas classes de NAT apresentadas.

Outro enfoque dado à metodologia de testes, visando sua simples reprodução e execução, foi o de utilizar o maior número possível de ferramentas e aplicações de uso gratuito ou publicamente disponíveis.

Foi adotada a virtualização para viabilizar a rápida implantação de todos os cenários de testes levantados, bem como viabilizar a reprodução futura destes testes em outras configurações de hardware. Além disso, a virtualização favorece o aspecto de ambiente controlado pois unifica hardware e minimiza imprevistos de rede entre os nós envolvidos em cada cenário.

6.2 Ambiente de Testes

O ambiente físico de testes consiste de um computador AMD Athlon 64 3200+ 2.20 GHz com 2 GB de memória RAM rodando Windows XP Service Pack 2 que foi usado como hospedeiro para as diversas máquinas virtuais linux usadas nos testes. A máquina hospederia também foi usada diretamente em alguns dos cenários de testes.

Em um dos cenários foi utilizado um outro computador AMD Athlon 64 3000+ 1.8 GHz com 1 GB de memória RAM rodando Windows XP Service Pack 2 usado como cliente STUN para um dos testes de carga realizados. A idéia foi comparar os testes realizados nas VMs com um ambiente mais tradicional de forma a validar o valor do uso da virtualização inclusive para testes de carga.

Foram criadas também 5 máquinas virtuais rodando a distribuição Linux Ubuntu Hardy Heron 8.04 alternate instaladas utilizando-se a opção de linha de comando apenas. Três destas máquinas virtuais são *multi-homed*, ou seja, possuem 2 placas de rede. Além disso, todas foram criadas com 256 mb de RAM e 1 GB de disco rígido.

Com isso, o ambiente de testes ficou configurado assim:

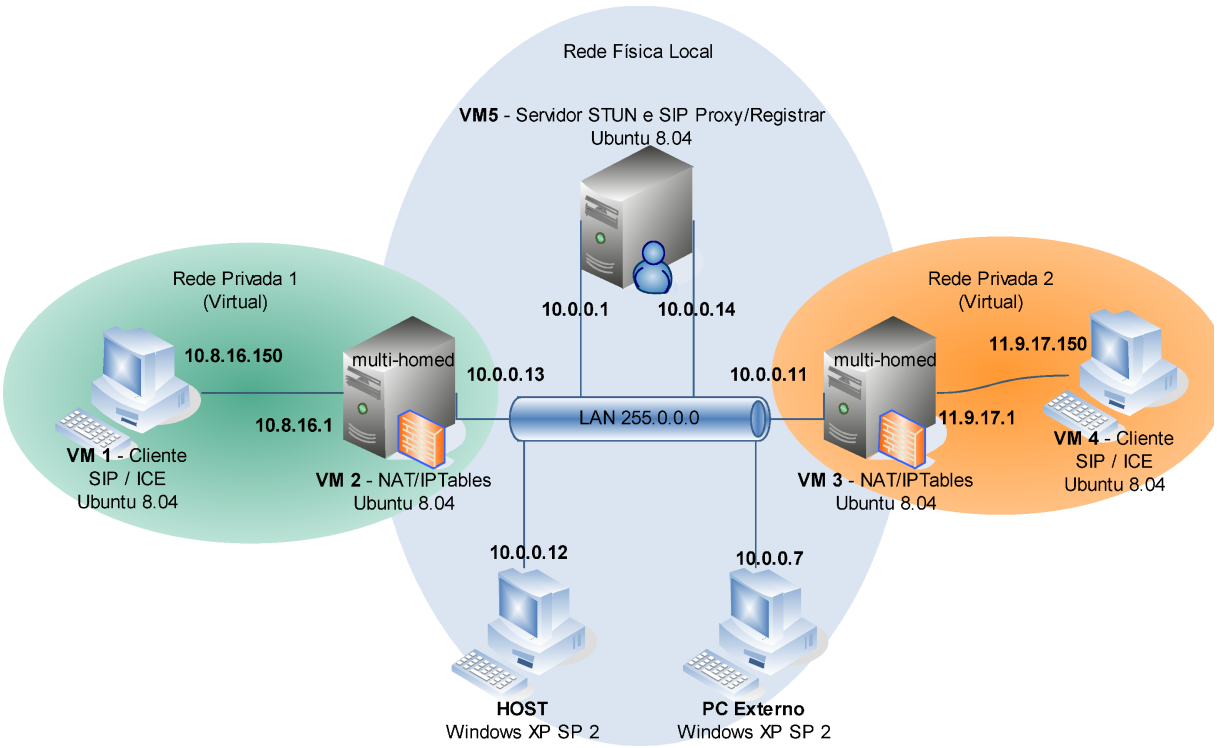


Figura 6.1: Ambiente de testes utilizando virtualização de nós e redes.

Em linhas gerais esse ambiente define três redes distintas. A rede central, em cinza, representa a rede física onde os dois computadores utilizados no cenário estão interligados, sendo uma rede privada doméstica comum. Para efeito de

cenário de testes esta rede representa a rede pública, assim como a Internet. As outras duas redes, em verde e laranja, são redes virtuais criadas pelo VMWare e representam nos cenários de testes as redes privadas que desejam se comunicar utilizando o ICE.

Nestas redes virtuais existe um nó isolado internamente em cada uma. Estas duas redes virtuais são interligadas a rede local (rede pública no cenário) através de um nó que faz o papel de gateway. Este gateway utiliza NAT dar acesso aos seus respectivos nós internos. Na rede local encontra-se um servidor SIP e um servidor STUN ambos plenamente acessíveis aos nós internos das redes privadas por meio de seus respectivos gateways (NAT).

As três redes devem possuir máscaras de subrede distintas, de forma a manter o tráfego limitado a seus participantes. Os gateways são nós *multi-homed* e participam tanto de suas respectivas redes privadas por uma de suas interfaces de rede como da rede publica através da outra interface de rede.

Nó	OS	Rede	Interface 1	Interface 2
HOST (Hospedeiro VM)	Windows XP	Física Local	10.0.0.12	-
Externo (Cliente STUN)	Windows XP	Física Local	10.0.0.7	-
VM 1 (Cliente ICE)	Ubuntu 8.04	Privada 1	10.8.16.150	-
VM 2 (Gateway)	Ubuntu 8.04	Privada 1/Local	10.0.0.13	10.8.16.1
VM 3 (Gateway)	Ubuntu 8.04	Privada 2/Local	10.0.0.11	11.9.17.1
VM 4 (Cliente ICE)	Ubuntu 8.04	Privada 2	11.9.17.150	-
VM 5 (STUN e SIP)	Ubuntu 8.04	Física Local	10.0.0.1	10.0.0.14

Tabela 6.1: Configurações do ambiente de testes

Então, de acordo com a Fig.6.1 e com a tabela 6.1 podemos identificar os elementos deste cenário:

- **Rede privada virtual 1** - Representada pelo balão verde, faz o papel de uma das redes privadas do cenário;
- **Rede privada virtual 2** - Representada pelo balão laranja, faz o papel da outra rede privada do cenário;
- **Rede Física Local** - Representada pelo balão cinza, faz o papel da rede pública do cenário;
- **VM 1** - Nó interno a rede privada 1;
- **VM 2** - Gateway de acesso da rede privada 1 à rede pública configurado com Iptables;
- **VM 3** - Gateway de acesso da rede privada 2 à rede pública configurado com Iptables;
- **VM 4** - Nó interno a rede privada 2;

- **VM 5** - Nó da rede pública onde encontram-se os servidores STUN e SIP Proxy/Registrar;
- **HOST** - Nó da rede pública que hospeda as máquinas virtuais e também atua como cliente STUN;
- **Externo** - Outro computador conectado a pública utilizado para validar testes de carga dos servidores STUN.

Para a realização dos testes e avaliação dos dados foram utilizadas as seguintes ferramentas:

6.2.1 PJSIP/PJNATH 0.8.0

É uma biblioteca utilitária de código aberto (GPL) que implementa o procolo SIP focando em alta performance, *small footprint*, para dispositivos embarcados ou não. Um dos recursos dessa biblioteca é a implementação do protocolo ICE através do subprojeto PJNATH.

Esta biblioteca foi utilizada como parâmetro de comparação para a validação dos pacotes STUN gerados pelo JICE e também para comparação dos testes de carga realizados no servidor STUN implementado pelo JICE.

Por ser implementada na linguagem C representa um bom comparativo de performance para os testes de carga com relação a linguagem Java e o modelo de programação de rede proposto pelo MINA.

Ela é também a única implementação disponível do STUN/ICE de acordo com suas versões mais recentes sendo o único candidato encontrado para a realização de testes de equivalência e comparativos de performance com o JICE.

6.2.2 Sun Java JRE VM 1.6

Máquina virtual Java implementada pela Sun versão 1.6. Conforme foi dito anteriormente foi dada preferência a última versão da máquina virtual pois ela oferece novos recursos para a programação em rede além de melhor performance.

Nas máquinas Windows foi utilizada a última versão disponibilizada no site da Sun. Nas máquinas Linux Ubuntu foi utilizada a versão 1.6 disponível na ferramenta de instalação de pacotes deste O.S.

6.2.3 VMWare Workstation 6.04 build 93057

O VMWare é um software proprietário e comercial que implementa a virtualização de hardware e é ideal para testar ambientes heterogêneos em redes com diversos

nós e plataformas. Ele permite reduzir bem os custos de implementação de um ambiente de testes de rede ao mesmo tempo que favorece um melhor controle deste ambiente.

Um recurso interessante oferecido por este produto e que foi incorporado a este projeto é a virtualização de interfaces de rede com a possibilidade de se criar diversas subredes privadas entre mais de uma máquina virtual.

6.2.4 Wireshark 1.0.0

O Wireshark é um software livre utilizado para a captura e análise de pacotes. Ele foi utilizado neste projeto, tanto na parte de testes como ao longo do desenvolvimento do software. Ele permite se analisar os pacotes e seu conteúdo e ainda elaborar estatísticas sobre este tráfego.

6.2.5 tshark 1.0.0

O tshark é a versão de linha de comando do wireshark e foi utilizado para a captação de pacotes nas máquinas virtuais, já que está não possuem modo gráfico. Os arquivos de dump gerados nas VMs foram transferidos para a máquina host e então analisados de forma central pelo a interface do Wireshark.

6.2.6 Netfilter/Iptables 1.3.8

O Iptables é uma ferramenta para filtro de pacotes tipo IPv4 baseado em regras. Ele pode ser usado como um firewall simples, por bloqueio de portas e protocolos, e principalmente para a implementação de NAT.

Ele foi utilizado para implementar o NAT de forma controlada pois como sua configuração é feita por regras é possível configurar os diversos comportamentos de um NAT através dele. Porém, ele não possui uma forma ampla de implementar os diversos tipos de NAT, apenas o tipo simétrico e de cone restrito por porta, os dois mais restritos. Os outros dois tipos têm de ser criados por regras estáticas através do encaminhamento de endereços e portas e pelo mapeamento estático da tradução destes endereços.

6.3 Metodologia e Cenários

Para cada objetivo estabelecido foram derivados os seguintes cenários e metodologias de teste:

6.3.1 Avaliação da carga de um servidor STUN

Analisando-se o protocolo ICE observa-se que na verdade é o servidor STUN quem deverá sofrer a maior carga em cenários reais. Isto porque o ICE é um protocolo em nível de aplicação cliente, então não se prevê cenários de grande carga para o mesmo. Além disso, o ICE, além de fazer uso de servidores STUN e TURN, ele próprio é uma extensão do STUN. Sendo assim, parece razoável que o objeto dos testes de carga seja exatamente o servidor STUN, avaliando-se o seu codificador e decodificador de mensagens, a capacidade em servir requisições simultâneas e a validade das respostas geradas.

Como base de comparação foi utilizado o servidor STUN disponível na biblioteca PJNATH. Para isso, esta foi compilada para o Ubuntu 8.04. Verificou-se que este servidor imprimia todas as mensagens recebidas e enviadas na tela, provavelmente para validação e testes, porém isso se mostrou um problema para uma efetiva comparação de carga. Em alguns testes, quando haviam muitas requisições simultâneas, esta característica gerava uma sobrecarga muito grande neste servidor. Como solução, retirou-se o código que realizava esta impressão em tela e o código foi recompilado, gerando assim um servidor bem mais eficiente e preparado para ser usado como base de comparação nos testes de carga. Logo o código do PJNATH utilizado neste cenário foi customizado para ser mais eficiente.

Em todos os cenários do teste de carga os servidores STUN se encontram na máquina VM 5, que utiliza o Ubuntu 8.04 como sistema operacional e possui a versão 1.6 da vm java instalada pelo gerenciador de pacotes. A variação dos três cenários avaliados se dará pela máquina onde será executado o aplicativo cliente responsável por gerar a carga nestes servidores.

6.3.1.1 Equivalência funcional das implementações

Antes da realização dos testes de carga foi feita uma verificação dos pacotes retornados por cada servidor para um mesmo tipo de requisição. Observa-se pela figura 6.2 que ambas implementações são bem equivalentes, sendo que o JICE relaciona o atributo XORMappedAddress antes do atributo MappedAddress visto que este deverá ser usado preferencialmente, no PJNATH ocorre o contrário, provavelmente como forma de otimizar o processamento do cliente ao receber a resposta. Esta ordem têm menos importância em termos práticos mas pode ser utilizada aqui para identificar as diferentes implementações, pois em todos os outros aspectos as mensagens são idênticas em suas estrutura, como era de se esperar.

Vale notar também que o Wireshark já reconhece uma mensagem com a estrutura da segunda versão do protocolo STUN e a classifica como stun2. Ele é capaz de decompor seus atributos e inclusive retirar a máscara xor do endereço mapeado no caso do atributo XORMappedAddress.

```
No.     Time        Source                Destination           Protocol Info
      4 0.001570    10.0.0.1              10.0.0.12             STUN2    Binding Succes
s Response

Frame 4 (94 bytes on wire, 94 bytes captured)
Ethernet II, Src: Vmware_a7:f9:1f (00:0c:29:a7:f9:1f), Dst: AsustekC_80:b0:e3 (00:11:d8
:80:b0:e3)
Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.12 (10.0.0.12)
User Datagram Protocol, Src Port: nat-stun-port (3478), Dst Port: rich-cp (2057)
Session Traversal Utilities for NAT
  .... ...1 ...0 .... = Message Class: Success Response (0x0010)
  ..00 000. 000. 0001 = Message Method: Binding (0x0001)
  Message Length: 32
  Message Cookie: 2112A442
  Message Transaction ID: AD063C6BE879384DED4E9E4B
  Attributes
    Attribute: MAPPED-ADDRESS
      Attribute Type: MAPPED-ADDRESS (0x0001)
      Attribute Length: 8
      Protocol Family: IPv4 (0x0001)
      Port: 2057
      IP: 10.0.0.12 (10.0.0.12)
    Attribute: XOR-MAPPED-ADDRESS
      Attribute Type: XOR-MAPPED-ADDRESS (0x0020)
      Attribute Length: 8
      Protocol Family: IPv4 (0x0001)
      Port (XOR-d): 10523
      [Port: 2057]
      IP (XOR-d): 43.18.164.78 (43.18.164.78)
      [IP: 10.0.0.12 (10.0.0.12)]
    Attribute: FINGERPRINT
      Attribute Type: FINGERPRINT (0x8028)
      Attribute Length: 4
      CRC-32: 0x036ca9bd

No.     Time        Source                Destination           Protocol Info
      8 28.755696   10.0.0.1              10.0.0.12             STUN2    Binding Succes
s Response

Frame 8 (94 bytes on wire, 94 bytes captured)
Ethernet II, Src: Vmware_a7:f9:1f (00:0c:29:a7:f9:1f), Dst: AsustekC_80:b0:e3 (00:11:d8
:80:b0:e3)
Internet Protocol, Src: 10.0.0.1 (10.0.0.1), Dst: 10.0.0.12 (10.0.0.12)
User Datagram Protocol, Src Port: nat-stun-port (3478), Dst Port: icg-bridge (2063)
Session Traversal Utilities for NAT
  .... ...1 ...0 .... = Message Class: Success Response (0x0010)
  ..00 000. 000. 0001 = Message Method: Binding (0x0001)
  Message Length: 32
  Message Cookie: 2112A442
  Message Transaction ID: DAF7C4DE2FD3DEF361B9C49B
  Attributes
    Attribute: XOR-MAPPED-ADDRESS
      Attribute Type: XOR-MAPPED-ADDRESS (0x0020)
      Attribute Length: 8
      Protocol Family: IPv4 (0x0001)
      Port (XOR-d): 10525
      [Port: 2063]
      IP (XOR-d): 43.18.164.78 (43.18.164.78)
      [IP: 10.0.0.12 (10.0.0.12)]
    Attribute: MAPPED-ADDRESS
      Attribute Type: MAPPED-ADDRESS (0x0001)
      Attribute Length: 8
      Protocol Family: IPv4 (0x0001)
      Port: 2063
      IP: 10.0.0.12 (10.0.0.12)
    Attribute: FINGERPRINT
      Attribute Type: FINGERPRINT (0x8028)
      Attribute Length: 4
      CRC-32: 0x59749fe7
```

Figura 6.2: Mensagens de resposta Binding do PJNATH e do JICE respectivamente, de acordo com captura realizada pelo Wireshark.

6.3.1.2 Cliente utilizado para criação da carga nos servidores

A metodologia de testes para este cenário compreende a realização de requisições concorrentes por um cliente de forma a gerar um carga no servidor STUN sendo avaliado. Como o protocolo STUN é estruturalmente bem simples e não requer, ao menos para o uso Binding, que seja guardado estado ele se mostrou um protocolo rápido e eficiente. Como efeito, nos testes onde não haviam requisições suficientes para estressar o servidor em geral o tempo de execução ficou limitado a eficiência do agente cliente em enviar requisições.

As requisições foram enviadas sempre por um mesmo agente cliente:

br.com.socia.jice.stun.usage.bind.test.LoadTestClientMain

Este agente, implementado em Java utilizando o cliente STUN do JICE, foi utilizado para testar ambos os servidores, JICE e PJNATH. Seu funcionamento se dá através do disparo de x *threads* cada uma realizando y transações Binding de forma sequencial com o servidor STUN em questão. Cada transação obedece o ciclo de vida para uma transação cliente do protocolo STUN podendo haver reenvios de acordo com o regime de *timeouts*.

Sua configuração se dá pelos seguintes parâmetros ip do servidor stun, número de *threads* disparadas e número de transações executadas por cada *thread*. O número de *threads* multiplicado pelo número de transações resulta no número de transações realizadas com o servidor. Ao final da execução este agente imprime o tempo em milesegundos(ms) gasto para realizar todas as requisições. Este tempo foi relacionado ao número de requisições e então comparado entre cada implementação, de acordo com o cenário utilizado.

Em todos os cenários foi utilizada as seguintes configurações para o agente cliente responsável por originar a carga:

Servidor STUN	Número de *Threads*	Número de Transações	Total de Transações
10.0.0.1	1	50	**50**
10.0.0.1	10	50	**500**
10.0.0.1	100	50	**5000**
10.0.0.1	200	50	**10000**
10.0.0.1	200	200	**40000**

Tabela 6.2: Configurações do agente cliente para os testes de carga

6.3.1.3 Testes de carga utilizando a máquina HOST

Neste cenário foi utilizada a máquina HOST para originar as requisições. Os servidores, conforme foi dito, se encontram na VM 5 e esta também está sendo executada na máquina HOST. Então neste cenário, os recursos da máquina HOST vão estar divididos entre o cliente que vai originar a carga e o servidor que irá

servir a este cliente.

Neste cenário obtivemos os seguintes resultados:

Requisições	PJSIP (C) (s)	JICE (Java/MINA) (s)
50	5,737702138	5,744770634
500	5,802881093	5,818012624
5000	6,536387446	6,441485998
10000	8,072423171	7,725717375
40000	23,09711039	23,19645905

Tabela 6.3: Resultado do teste carga originado na máquina HOST.

Pode-se perceber que enquanto não houveram requisições suficientes para utilizar todos os recursos da máquina que os valores ficaram próximos do patamar mínimo, que é o tempo de latência envolvido no acesso a rede em cada requisição e ainda pelo fato das transações serem executadas serialmente em cada thread.

No momento que os recursos de processamento e memória da máquina foram completamente utilizados o tempo de execução aumentou significativamente (fig. 6.3). Porém neste caso não há como ter certeza se a maior parte do tempo está sendo gasta com servidor o STUN propriamente ou com a gerência das diversas threads clientes.

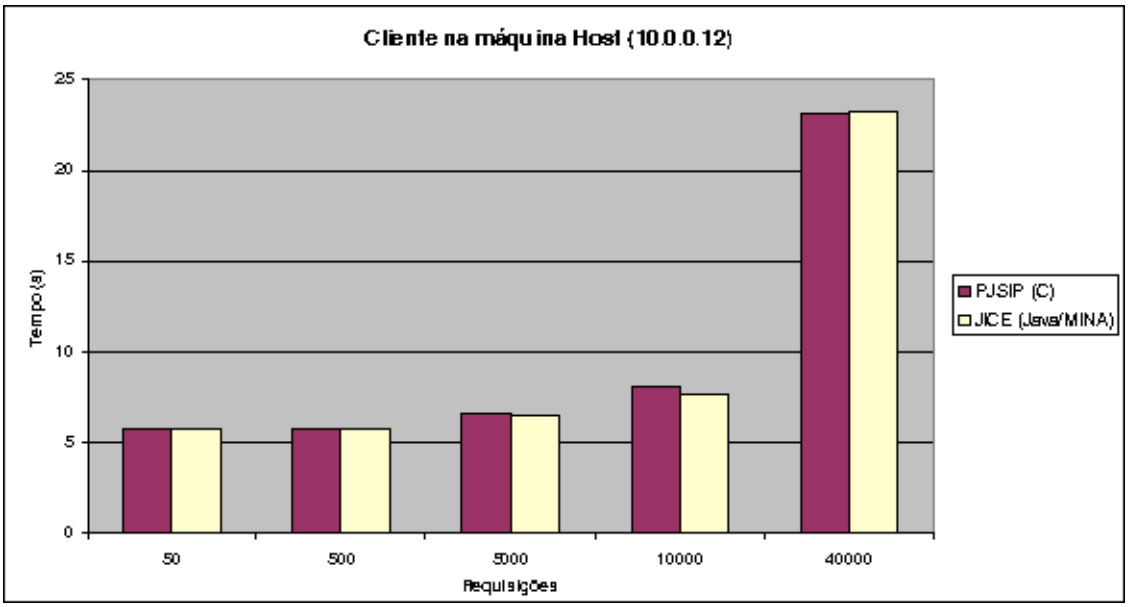


Figura 6.3: Teste de carga com o cliente STUN na máquina HOST.

6.3.1.4 Testes de carga em ambiente com NAT

Com o objetivo de avaliar a carga em cenários onde há de fato um NAT envolvido optou-se por utilizar o cliente STUN na máquina VM 1. Assim, sua requisição

pelo método Binding vai de fato retornar um endereço reflexivo diferente do seu endereço local, retratando exatamente o mapeamento (Binding) realizado pelo NAT para aquela transação.

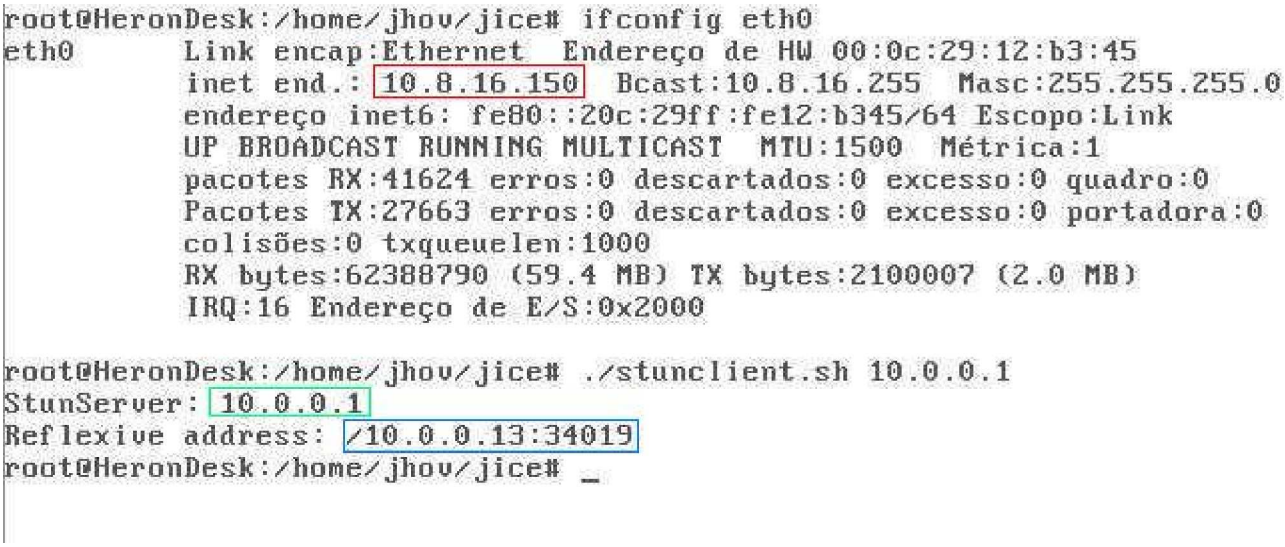


Figura 6.4: Comparação entre o endereço local de um nó e o mapeamento realizado pelo NAT e descoberto pelo método Binding do protocolo STUN.

Na figura 6.4 observa-se em vermelho o endereço associado a interface do nó VM 1. Em verde encontra-se o endereço do servidor STUN utilizado para revelar o Binding criado pelo NAT para este nó. Nota-se que o endereço do servidor STUN somente é acessível e roteável por intermédio do *gateway* (VM 2) associado a este nó. O endereço reflexivo descoberto através do STUN encontra-se em azul.

Neste cenário obtivemos os seguintes resultados:

Requisições	PJSIP (C) (s)	JICE (Java/MINA) (s)
50	5,147460807	5,143542924
500	5,587833451	5,618912127
5000	11,21622692	8,577971046
10000	17,07123132	15,00872589
40000	51,43022604	52,55934947

Tabela 6.4: Resultado do teste de carga em ambiente com NAT (VM 1).

Neste cenário observa-se um grande aumento do tempo de execução em comparação ao cenário anterior. Em primeiro momento isso foi atribuído a existência de NAT e ao fato deste cenário operar três máquinas virtuais ao mesmo tempo. Por outro lado este impacto só foi sentido quando os recursos da VM onde rodava o cliente ficaram escassos. Sendo assim é mais provável que este salto no tempo de execução (fig. 6.5) se deva a quantidade limitada de recursos desta VM, em

especial a memória disponibilizada para a execução da máquina virtual Java que foi de 128 mb, metade do valor utilizado nos outros cenários.

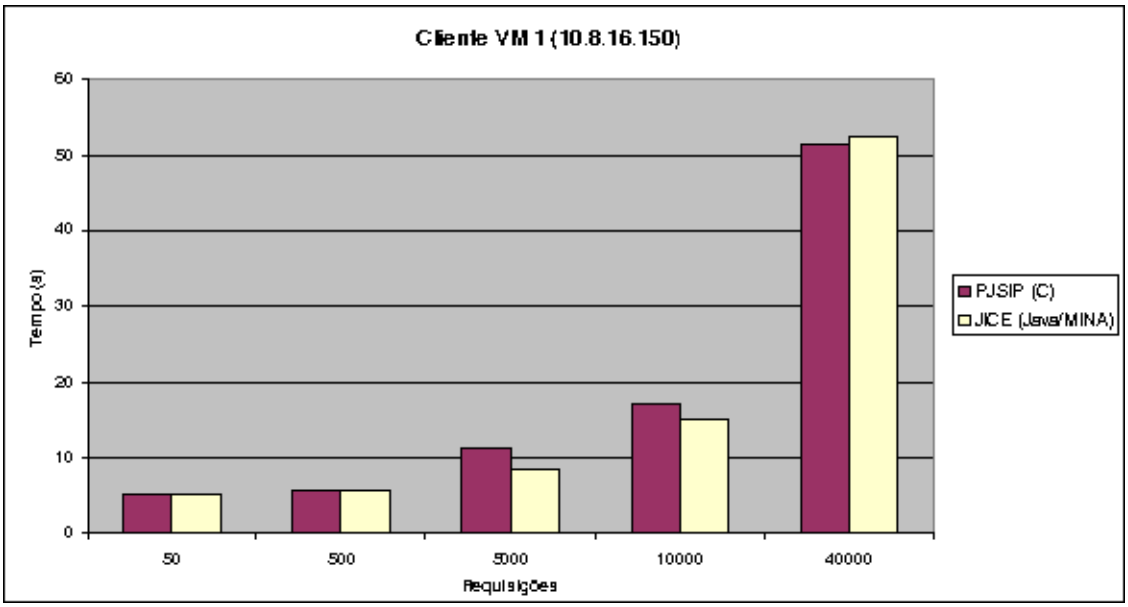


Figura 6.5: Teste de carga em ambiente com NAT (VM 1).

6.3.1.5 Testes de carga utilizando um outro computador externo

Para avaliar melhor o impacto de se utilizar a virtualização, inclusive para testes de carga onde ambos os clientes e servidores rodavam em um mesmo computador monoprocessado, optou-se por utilizar um segundo computador, chamado Externo, como cliente para originar a carga para os servidores STUN.

Esta máquina externa está conectado a máquina HOST através da rede local física, logo este teste foi realizado em uma infraestrutura real de rede e cabeamento, logo representa um ambiente menos controlado e mais real. Espera-se que o consumo de recursos pelo cliente não afete a execução dos servidores, logo os diferentes modelos de gerência de memória das plataformas em questão não deverão influenciar de forma significativa estes testes como pode ter ocorrido nos dois cenários anteriores.

Neste cenário obtivemos os seguintes resultados:
Este cenário se mostrou bastante interessante. Ao mesmo tempo em que ele não superou os dados do primeiro cenário, como se esperava que o fizesse, ele apresentou dados mais significativos para a comparação entre as duas implementações.

No primeiro caso atribui-se ao fato de ele estar usando um cenário real de rede, ou seja o tráfego está realmente sendo serializado e trafegando por cabeamento e hubs, e isso aumentou a latência de cada transação aumentando seu tempo de execução. Isto pode ser notado inclusive nos testes com menos requisições, o que

Requisições	PJSIP (C) (s)	JICE (Java/MINA) (s)
50	5,851646787	5,840802672
500	5,940414494	5,937116589
5000	7,477119298	7,034257706
10000	11,67760958	9,790712227
40000	32,03674601	27,41903147

Tabela 6.5: Teste de carga com uma segunda máquina como cliente (PC Externo).

reforça esta hipótese.

Por outro lado, como os recursos da máquina HOST neste cenário foram todos destinados aos servidores STUN (VM 5) pode-se ter um retrato mais fiel do comparativo entre as duas implementações. De acordo com a figura 6.6 pode-se notar uma diferença mais significativa dos tempos de execução em favor do JICE. Isso demonstra que havendo recursos suficientes a implementação Java/MINA pode apresentar resultados muito favoráveis podendo até superar soluções ditas nativas. Isso se deve ao modelo assíncrono utilizado pelo MINA e principalmente pelas boas práticas e técnicas incorporadas pelo mesmo nos aplicativos que dele fazem uso, assim como o JICE.

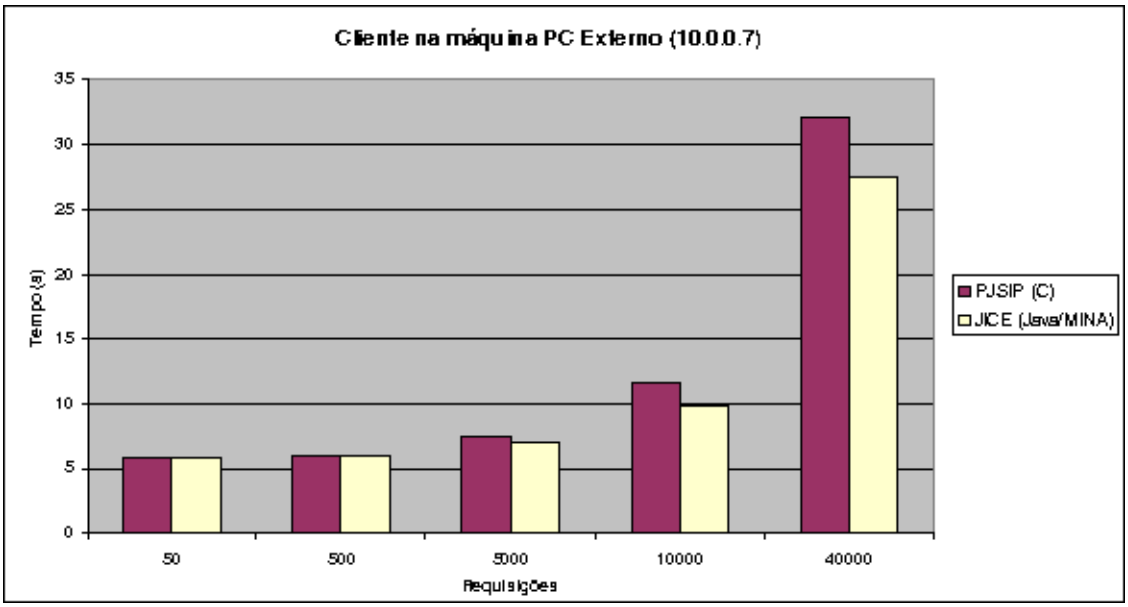


Figura 6.6: Teste de carga em ambiente com uma segunda máquina como cliente (PC Externo).

6.3.2 Avaliação da travessia de NAT em ambiente controlado

O objetivo deste cenário foi criar um ambiente controlado para realizar testes de conectividade entre as 4 classes de NAT. Este ambiente controlado foi realizado

exatamente pelo ambiente de testes descrito anteriormente.

Considerando que existem 4 classes de comportamento para o NAT existem 10 possibilidades de configurações diferentes para este cenário. Essas possibilidades consideram uma classe interagindo com todas as outras e também com ela mesma.

Assim sendo, para cada possibilidade de configuração de ambos os gateways com NAT, tentou-se estabelecer um fluxo multimídia entre os dois nós internos fazendo uso do ICE.

Os resultados obtidos neste cenário foram apurados da seguinte forma. Para cada combinação de comportamentos o resultado é satisfatório se houver comunicação efetiva, ponto a ponto, entre os nós tentando estabelecer a comunicação, ou seja se de fato ocorrer a travessia de NAT.

As seguintes combinações foram testadas com relação a conectividade utilizando-se combinações de NAT configurados nos gateways (VM 2 e VM 3), partindo das mais permissivas para as menos permissivas:

Tipo de NAT (VM 2)	Tipo de NAT (VM 3)	Conectividade P2P
Cone Total	Cone Total	**Efetiva**
Cone Total	Cone Restrito	**Efetiva**
Cone Total	Cone Restrito por porta	**Efetiva**
Cone Total	Simétrico	**Efetiva**
Cone Restrito	Cone Restrito	**Efetiva**
Cone Restrito	Cone Restrito por porta	**Efetiva**
Cone Restrito	Simétrico	**Efetiva**
Cone Restrito por porta	Cone Restrito por porta	**Efetiva**
Cone Restrito por porta	Simétrico	**Nula**
Simétrico	Simétrico	**Nula**

Tabela 6.6: Avaliação da conectividade de acordo com o tipo de NAT.

Segundo a tabela 6.6, pode-se verificar que o ICE apresenta grande eficácia em estabelecer conexões P2P efetivas entre a maioria das classes de NAT existentes, ao menos no cenário controlado, que por ter sido criado ao pé da letra, muitas vezes com regras estáticas no IPTables, não retrata necessariamente a realidade. Felizmente a distribuição de NAT por estas classes favorece, ao que parece, os tipos menos restritos, principalmente o tipo Cone Restrito.

Por outro lado, as redes corporativas estão cada vez mais populares e estas possuem regras mais rígidas, em geral utilizando firewalls e gateways. Os tipos de classe predominantes neste cenário são Cone Restrito por porta e principalmente o Simétrico pela popularização do uso de gateways Linux.

Enquanto este fator sozinho não impede a conectividade ele a limita, em especial em fluxos inter-corporações, usuários importantes de tecnologias como VoIP

e *streaming* de vídeo.

Faz-se necessário uma pesquisa quantitativa extensiva para determinar a real proporção de cada um dos tipos de NAT. Essas pesquisas deverão ser conduzidas periodicamente para avaliar as tendências das implementações de NAT relacionando estas tendências com o processo de exaustão do endereçamento IPv4 e adoção do IPv6.

Atualmente existem algumas diretivas para tornar o NAT menos restrito com relação a conectividade sem contudo expor suas redes a riscos. Resta saber se os fabricantes estão dispostos a criarem produtos segundo estes padrões.

Um ponto positivo observado neste teste de conectividade foi notar como o uso da virtualização, de subredes especialmente, facilitou e viabilizou os testes aqui realizados. Ela se mostrou extremamente estável e confiável, sem engessar o uso das ferramentas de verificação dos dados. A única restrição é que estes ambientes em geral são mais limitados em recursos, logo não foi possível instalar um gerenciador de janelas nas máquinas virtuais com Linux. Neste caso, a captura de pacotes, quando necessária, foi realizada através da ótima ferramenta de terminal disponibilizada pela equipe do Wireshark, o Tshark.

6.4 Avaliação de resultados

Durante os testes buscou-se avaliar o JICE em termos quantitativos e qualitativos. Em especial foi realizado um comparativo com a biblioteca PJNATH avaliando-se a equivalência entre estas soluções e sua eficiência relativa.

Verificou-se em ambos os tipos de testes que as mensagens STUN geradas pelo JICE são equivalentes em estrutura e valores as mensagens da biblioteca PJNATH, sugerindo que ele de fato é uma implementação fiel do protocolo ICE. Inclusive ambas as bibliotecas se mostraram absolutamente interoperáveis, em nível cliente e servidor.

6.4.1 Testes de carga

Percebeu-se que o STUN é um protocolo muito leve e logo requer uma enorme quantidade de requisições para ser avaliado. Elaborar um cliente para gerar este tipo de carga se mostrou um tarefa difícil, em especial quando ele é executado em uma única máquina. Isso porque ele consome muitos recursos de memória principalmente relativo aos canais de rede alocados. Não existem muitas portas disponíveis para comunicação com o servidor e isso se mostrou um gargalo.

Para contornar estes problemas foram utilizadas *threads* na programação do cliente que gerou a carga para os testes dos servidores. Com isso pode-se utilizar ao máximo os recursos de processamento da máquina em questão, pois enquanto

uma transação estava bloqueada aguardando uma leitura ou escrita de rede, as outras podiam continuar sua execução normalmente.

Ainda assim, os testes só se mostraram significativos quando os recursos da máquina eram todos utilizados, pois ai sim este modelo de programação por *threads* pode revelar sua utilidade.

Pode-se notar que o uso de um ambiente controlado utilizando-se virtualização representa uma forma viável e eficiente também para a realização de testes de carga. Entretanto, sugere-se a utilização de máquinas multiprocessadas para evitar-se distorções que possam ocorrer pelo compartilhamento de recursos.

Um ponto a ser avaliado é o porque dos dados comparativos entre o JICE e o PJNATH foram atenuados neste tipo de ambiente. Um hipótese é que a menor latência no acesso a rede possa ter compensado algumas deficiências e talvez seja exatamente as rotinas de leitura e escrita de dados em rede o maior diferencial da abordagem dada pelo JICE/MINA através do modelo assíncrono. Outro ponto a ser explorado é se a constante troca de contexto em máquinas monoprocessadas também podem levar a este tipo de atenuação.

De forma geral as implementações avaliadas apresentaram resultados muito semelhantes, estatisticamente equivalentes. O JICE se mostrou um pouco mais eficiente em termos de tempo de execução com relação ao PJNATH. Esse tópico ainda pode ser melhor explorado através de uma abordagem mais completa, fazendo-se uma análise da evolução do consumo de recursos, em especial a memória, visto que o modelo de gerência deste recurso é bem diferente em cada plataforma.

6.4.2 Testes funcionais de travessia

Com relação aos testes de conectividade entre as diversas classes de NAT pode-se obter um alto grau de efetividade nas transações avaliadas. Ainda que este ambiente reproduza fielmente os comportamentos definidos pelas classes de NAT. Logo, o teste pode ter sido eficaz apenas porque o cenário foi criado para isso e não incorpora as exceções existentes em NAT "reais".

Ainda assim, pode-se perceber como o protocolo é robusto, pois viabilizou a comunicação mesmo em casos onde apenas um dos nós se encontrava em uma rede com um NAT mais permissivo.

Vale destacar o uso da virtualização nestes testes. Ela viabilizou a criação e fácil reprodução deste cenário reduzindo custos do ambiente e também tempo de implantação. Se mostrou uma solução efetiva para testes em ambientes de rede heterogêneos e complexos.

Capítulo 7

Conclusão

Este trabalho teve por objetivo realizar a implementação do protocolo ICE. Para isso realizou a contextualização da problemática do NAT, suas origens, suas características, suas classificações e como o NAT pode afetar a conectividade entre dois nós quebrando o modelo p2p original do IPv4.

Depois disso procurou detalhar as possíveis soluções para o problema, mostrando que até o ICE ainda não havia uma solução geral e padronizada para este problema. Mostrou que apenar do IPv6 se apresentar como uma solução definitiva para esta questão seu processo de transição será gradual necessitando inclusive de soluções de NAT durante a transição.

Ou seja, este é um problema atual, importante, que deverá durar por alguns anos ainda. Logo, a necessidade de um protocolo que padronize e ofereça uma solução efetiva para a travessia de NAT.

Então foi feito um estudo detalhando este protocolo e suas associações. Considerando que o ICE é um protocolo extenso e que incorpora outros protocolos extensos este estudo se mostrou muito trabalhoso e demorado. Ainda assim, considera-se que o principal objetivo de entender o funcionamento do protocolo e seu ciclo de vida foi alcançado.

Neste contexto foi feita a implementação do NAT na plataforma Java, pois esta ainda não possuia tal instrumento. Esta implementação fez uso da biblioteca Apache MINA para incorporar o modelo não blocante das novas APIs de programação em redes disponibilizados pela nova VM Java. Com esta framework foi criado as bases para o protocolo STUN, estrutura das mensagens e processos de envio e recebimento.

A avaliação desta implementação mostrou resultados muito favoráveis em um comparativo com outra implementação do ICE em C, o PJNATH. O servidor STUN criado pelo JICE foi mais eficiente na maioria dos casos e apresentou um completo nível de equivalência com relação as mensagens STUN geradas. Foi possível inclusive realizar a interoperabilidade entre as implementações possibilitando chamar o servidor de uma com o cliente da outra e vice-versa.

Assim, o uso do Apache MINA se mostrou uma opção muito interessante para a programação de aplicativos de rede focados em alta performance. Além disso, sua arquitetura bem definida e apoiada em boas práticas de programação bem como seu modelo baseado em eventos ajudou a reduzir consideravelmente a complexidade e inteligibilidade do código produzido. Em especial quando se compara o código em Java/MINA com o de seu par avaliado PJNATH.

Com relação a conectividade pode-se notar que o ICE é um protocolo elegante e robusto. Ele se apresenta como um protocolo maduro e bem elaborado para se encaixar de forma muito direta aos protocolos que utilizem o modelo de Oferta e Procura.

Mais ainda, seu método se mostrou muito eficiente em estabelecer a conectividade nos cenários avaliados atingindo 80% de sucesso entre as possibilidades existentes.

O uso da virtualização para a implantação do ambiente de testes se mostrou muito robusto e eficaz. Além de oferecer um ambiente maleável e de simples configuração ela pode isolar os testes dos imprevistos e inconstâncias usualmente encontrados em ambientes reais de testes. Foi possível também tirar algumas conclusões a respeitos dos protocolos e suas implementações assim como sugestões de pesquisas e trabalhos futuros.

7.1 Dificuldades encontradas

A maior dificuldade encontrada foi a de administrar a crescente complexidade na avaliação e implementação do protocolo ICE. Como ele estende e se acopla a diversos outros protocolos, estes outros que por sua vez fazem o mesmo, fez com que esse projeto se tornasse um jogo de gato e rato em uma perseguição sem fim pelo entendimento dos cenários, aplicações e inter-relações existentes entre os diversos participantes.

Além disso, a falta de outras implementações disponíveis e a baixa qualidade da documentação de algumas bibliotecas utilizadas também foram um grande desafio a ser transposto. Em especial, as bibliotecas PJNATH e jain-sip são muito pouco documentadas e como contra-ponto se apresentam como bibliotecas de grande complexidade.

Outra dificuldade encontrada foi a programação em rede utilizando-se Java, em especial com relação ao uso de tipos primitivos que obrigatoriamente utilizam o bit mais significativo para propósitos de sinalização. Porém, depois que este problema foi compreendido ele foi rapidamente mitigado e então o uso do Apache MINA trouxe uma simplicidade muito grande para a programação nesta plataforma.

A criação de uma *stack* SIP simples capaz de acoplar o JICE e ao mesmo tempo viabilizar cenários completos de testes também foi um desafio muito grande a ser vencido. Em especial porque a biblioteca jain-sip resolve o SIP em um nível muito baixo, ela mantém este protocolo o mais geral possível, o que torna complexo e muito sujeito a falhas o desenvolvimento de aplicativos utilizando-a. Em especial se o desenvolvedor não tiver um conhecimento profundo sobre a estrutura de uma mensagem SIP e seus diversos atributos.

Outra questão com relação a implementação da stack SIP foi a criação de entidades que possibilitassem a solução JICE ser testada em cenários reais via Internet. É que para estes cenários é necessário implementar um SIP Proxy para contornar o NAT no nível SIP e também um SIP Registrar para registrar novos usuários e divulgá-los para os outros usuários ativos para fins de conexão. O jain-sip possui implementações de ambas as entidades, porém estas estão inativas há algum tempo e estavam inclusive desatualizadas com relação ao core da biblioteca. Logo, implementar uma Stack SIP simples minimalista para integrar o JICE foi uma tarefa muito difícil e que mesmo assim não foi implementada como desejado.

7.2 Trabalhos Futuros

7.2.1 Evolução do JICE

Existem diversas atividades que precisam ser realizadas no projeto JICE para que ele se torne maduro efetivamente. Exatamente por isso este será publicado como código aberto, espera-se que a comunidade de software livre possa contribuir para sua evolução e validação.

Em especial o protocolo TURN deve ser implementado, pois ele é a garantia que o JICE poderá funcionar em 100% dos casos, mesmo aqueles onde a conectividade direta não for possível.

Seria interessante também realizar um *refactoring* que possibilitasse a substituição do Apache MINA por outras bibliotecas de programação em rede como o Grizzly. Isto porque, ainda que o Apache MINA seja extremamente eficiente e simples, podem existir outras soluções SIP que precisem integrar o JICE mas não podem utilizar uma outra forma de acesso aos recursos de rede.

Além de poder substituir o Apache MINA através do uso de interfaces seria importante criar uma versão que não utilizasse biblioteca alguma, seria java puro mesmo, talvez inclusive retrocedendo para o método anterior de programação da Java VM. Isto possibilitaria a portabilidade do JICE para as plataformas anteriores ou até para plataformas mais restritas como a dispositivos móveis e embarcados.

7.2.2 Novas linhas de pesquisa e desenvolvimento

Existem diversos cenários que ainda precisam ser avaliados para o JICE, o mais importante deles sendo o seu uso em cenários reais, sendo usado em aplicações efetivas de fluxos multimídias. Em especial, o JICE poderia ser portado para se integrar ao projeto sip-communicator que está produzindo um cliente sip para a plataforma Java. Este projeto ainda não têm uma solução ICE incorporada e este é um requisito muito aguardado segundo seu planejamento de atividades a serem realizadas.

Outros cenários que deveriam ser melhor avaliados são os que envolvem nós multi-homed (WiFI e LAN) e dual-stack (IPv4 e IPv6). Pois o método do ICE oferece recursos muito eficientes para lidar com estes cenários sempre em busca do melhor canal de comunicação.

Por fim, outras variáveis envolvidas ao longo da execução do servidor STUN em cenários de carga poderiam ser melhor analisadas, em especial a evolução do uso da memória e como as estratégias de *Garbage Collector* podem afetar suas funções.

Apêndice A

Lista de Acrônimos

3GPP	*3rd Generation Partnership Project*
API	*Application Programming Interface*
CIC	*Departamento de Ciência da Computação*
CRC	*Cyclic Redundancy Check*
FIFO	*First In First Out*
GPL	*GNU Public License*
HTTP	*HyperText Transfer Protocol*
ICE	*Interactive Connectivity Establishment*
IE	*Instituto de Ciências Exatas*
IETF	*Internet Engineering Task Force*
IMS	*IP Multimedia Subsystem*
ID	*Identificador Único*
IP	*Internet Protocol*
IPv4	*Internet Protocol Version 4*
IPv6	*Internet Protocol Version 6*
ISO	*Internet Standard Organization*
JSR	*Java Specification Request*
MINA	*Multi-purpose Infrastructure for Network Applications*
ms	*Milisegundos*
MTU	*Maximum Transmission Unit*
NAT	*Network Address Translation*
OSI	*Open Systems Interconnection*
P2P	*Peer to Peer*
PSTN	*Public Switched Telephone Network*
QoS	*Qualidade de Serviço*

RFC	*Request For Comments*
RTO	*Retransmission Timeout*
RTP	*Real-time Transport Protocol*
RTT	*Round-Trip-Time*
SDP	*Session Description Protcol*
SIP	*Session Initiation Protcol*
STUN	*Session Traversal Utilities for NAT*
TCP	*Transmission Control Protocol*
TTL	*Time To Live*
TURN	*Traversal Using Relays around NAT*
UDP	*User Datagram Protocol*
UnB	*Universidade de Brasília*
UPnP	*Universal Plug and Play*
VM	*Virtual Machine*
VoIP	*Voz sobre IP*

Apêndice B

Glossário e definições

Agente: Entidade definida pelo protocolo SIP e utilizada no STUN e no ICE. Representa uma unidade computacional que implemente protocolos em nível de aplicação (SIP / STUN /ICE). Podem atuar como clientes em umas transações e/ou como servidores em outras.

Agente é um termo mais amplo que nó, pois pode compreender mais de um protocolo de transporte, mais de uma interface de rede e ainda diversos protocolos em nível de aplicação.

Um agente SIP pode ser um servidor (Proxy, Registrar e Presence) ou um agente em nível de usuário que é responsável por iniciar e responder a chamadas (Softphones e IpPhones).

Cliente STUN: Agente que envia requisições ou indicações e recebe respostas.

Endereço de transporte: A combinação entre um endereço IP e um número de porta (TCP ou UDP).

Endereço de Transporte Reflexivo: Endereço de transporte associado a um cliente no lado publico de um NAT.

Nó: Termo muito utilizado por protocolos em nível de transporte e rede, especificamente em TCP, UDP, IPv4 e IPv6. Representa um ponto de conexão em uma rede de computadores onde dados podem ser transmitidos, recebidos e encaminhados.

Servidor STUN: Agente que recebe requisições ou indicações e envia respostas.

Referências

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

[2] Y. Rekhter, R. Moskowitz, D. Karrenberg, G. Groot e E. Lear *Address Allocation for Private Internets.* RFC 1918, Fevereiro de 1996.

[3] P. Srisuresh, B. Ford e D. Kegel. *State of Peer-to-Peer(P2P) Communication Across Network Address Translators(NATs).* draft-ietf-behave-p2p-state-04 (Work In Progress), Setembro de 2007.

[4] Ed. F. Audet e C. Jennings. *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP.* RFC 4787, Janeiro de 2007.

[5] D. Senie. *Network Address Translator (NAT)-Friendly Application Design Guidelines.* RFC 3235, Janeiro de 2002.

[6] P. Srisuresh e M. Holdrege. *IP Network Address Translator (NAT) Terminology and Considerations.* RFC 2663, Agosto de 1999.

[7] J. Rosenberg e H. Schulzrinne. *An Offer/Answer Model with Session Description Protocol (SDP).* RFC 3264, Junho de 2002.

[8] M. Handley, V. Jacobson e C. Perkins. *SDP: Session Description Protocol.* RFC 4566, Julho de 2006.

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley e E. Schooler. *SIP: Session Initiation Protocol.* RFC 3261, Junho de 2002.

[10] J. Rosenberg, C. Huitema, R. Mahy, P. Matthews e D. Wing. *Session Traversal Utilities for (NAT) (STUN).* draft-ietf-behave-rfc3489bis-15 (work in progress), Maio de 2008.

[11] Jonathan Rosenberg. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).* draft-ietf-behave-turn-07 (work in progress), Maio de 2008.

[12] Jonathan Rosenberg. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols.* draft-ietf-mmusic-ice-19 (work in progress), Maio de 2008.

[13] S. A. Baset e H. Schulzrinne. *An analysis of the skype peer-to-peer internet telephony protocol*. Columbia University, New York, NY, Tech. Rep. CUCS-039-04, 2004.

[14] B. Ford, P. Srisuresh e D. Kegel. *Peer-to-Peer Communication Across Network Address Translators*. In Proceedings of the USENIX Annual Technical Conference (Anaheim, CA), Abril de 2005.

[15] S. Guha e P. Francis. *Characterization and Measurement of TCP Traversal through NATs and Firewalls*. Proceedings of the USENIX Annual Technical Conference (Anaheim, CA), 199-211, Abril de 2005.

[16] Jeffrey L. Eppinger. *TCP Connections for P2P Apps: A Software Approach to Solving the NAT Problem*. Technical Report CMU-ISRI-05-104, Carnegie Mellon University, Janeiro de 2005.

[17] T. Wallingford. *Switching to VoIP*. O'Reilly, Junho de 2005.

[18] A. Tanenbaum. *Redes de Computadores*. Campus, 2003.

[19] T. Lee. *Mina Performance Test Reports*. http://mina.apache.org/performance-test-reports.html, acessado em 27/05/2008.

[20] P. Royal *Building TCP/IP Servers with Apache MINA*. ApacheCon EU 2007, http://mina.apache.org/documentation.data/ACEU2007.pdf, acessado em 27/05/2008.

[21] D. E. Comer *Internetworking with TCP/IP, Volume 1*. Prentice Hall, Quinta Edição, Junho de 2005.

[22] Sun Microsystems *The Java Tutorials*. http://java.sun.com/docs/books/tutorial/, última atualização em 14/03/2008.

[23] P. O'Doherty e M. Ranganathan *JAIN SIP API Specification*. http://jcp.org/aboutJava/communityprocess/mrel/jsr032/index.html, Novembro de 2006.

[24] K. R. Porter *SDP API*. http://jcp.org/aboutJava/communityprocess/pfd/jsr141/index.htm Julho de 2004.

[25] E. Proulx *An Introduction to the JAIN SIP API*. http://dev2dev.bea.com/pub/a/2007/10/introduction-jain-sip.html?page=1, acessado em 07/06/2008.

[26] IANA *Number Resources*. http://www.iana.org/numbers/, acessado em 17/05/2008.

[27] G. Houston *IPv4 Address Report*. http://www.potaroo.net/tools/ipv4/index.html, acessado em 30/05/2008.

[28] *IPv6 Forum.* http://www.ipv6forum.org/, acessado em 30/05/2008.

[29] G. Houston *An Update on IPv6 Deployment.* RIPE 56 Meeting, http://www.ripe.net/ripe/meetings/ripe-56/presentations/, Maio de 2008

[30] K. S. Evans (Adm. Office of E-Government and Information Technology) *MEMORANDUM FOR THE CHIEF INFORMATION OFFICERS.* http://www.whitehouse.gov/omb/memoranda/fy2005/m05-22.pdf, Agosto de 2005

[31] I. Beijnum *Address per country.* http://www.bgpexpert.com/addressespercountry.php, acessado em 30/05/2008

[32] Wikipedia *IPv6 Transition mechanisms.* http://en.wikipedia.org/wiki/IPv6#Transition_mechanisms, acessado em 30/05/2008