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

Avaliação do PCP como mecanismo de travessia de NAT em aplicações de Backup

Raphael Rodrigues Pereira
Pedro Henrique Lima Ferreira

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

Orientador
Prof. Dr. Jacir Luiz Bordim

Brasília
2019

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

Avaliação do PCP como mecanismo de travessia de NAT em aplicações de Backup

Raphael Rodrigues Pereira
Pedro Henrique Lima Ferreira

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

Prof. Dr. Jacir Luiz Bordim (Orientador)
CIC/UnB

Prof. M.E. Lucas Rodrigues Costa
Universidade de Brasília

Prof. M.E. Luis Alberto Belem Pacheco
Universidade de Brasília

Prof. Dr. Edison Ishikawa
Coordenador do Bacharelado em Ciência da Computação

Brasília, 25 de Julho de 2019

Dedicatória

Raphael Rodrigues Pereira. Dedico este trabalho em primeiro lugar a Deus, Nossa Senhora, São José, Padre Pio, São Francisco e à Santa (Madre) Teresa de Calcutá de quem sou devoto. Eles ajudaram-me e sempre auxiliaram-me a manter a paciência, ajudaram-me a vencer espiritualmente e mentalmente para que eu pudesse sair vitorioso da graduação.

Dedico também especialmente à minha querida e amada namorada, Marina de Oliveira Sampaio, que esteve comigo em todos os momentos tristes, felizes, de estresse, de calmaria. Ela ajudou-me imensamente em todo esse processo de terminar as disciplinas e concluir o curso, indo estudar comigo na UnB várias vezes, ajudando-me em tudo o que eu precisasse, aconselhando-me, auxiliando-me a persistir nessa luta, me proporcionando sorrisos e alegrias, dando-me um propósito e um sonho pra minha vida, me dizendo palavras de amor e de carinho, fazendo com que eu pudesse me tornar uma pessoa mais tranquila em momentos de desespero e de dificuldades, momentos esses que eu passei durante todo o meu caminho pela universidade.

Gratulo também à minha família, que formaram-me para que eu pudesse me tornar essa pessoa que sou e que tenho muito orgulho de ser. Principalmente à minha mãe, Lucia Rosa Rodrigues Pereira e ao meu pai, Marcos Maurício Rodrigues Pereira, que sempre apoiaram-me, deram-me conselhos e ajudaram-me a continuar insistindo nessa caminhada, entendendo todas as dificuldades pelas quais eu passei e tentando prestar auxilío sempre que necessário. Aos meus irmãos, Matheus Maurício Rodrigues Pereira e Felipe Rodrigues Pereira. Aos meus avós, Maria Conceição Rosa e Sandoval Rodrigues dos Santos, que me aconselharam e me apioaram sempre que precisei e ao meu falecido tio-avô, José Eustáquio Rosa (Ti Taquim), que inspirou em vários aspectos o meu modo de viver.

Gratifico também à família da minha namorada, Maria Geisa de Oliveira Sampaio, Raimundo Nonato Sampaio, Marisa de Oliveira Sampaio e Maristela de Oliveira Sampaio, que também foram muito importantes nos últimos três anos da minha caminhada na Universidade, rezando e torcendo por mim, acolhendo-me com carinho com conselhos e palavras de fé, me presenteando com lanchinhos e visitas à Universidade. Nunca esquecerei dessas atitudes de amor.

Pedro Henrique Lima Ferreira. Dedico este trabalho primeiramente à minha família, principalmente à minha mãe, Clarice, ao meu pai, José Antônio, e à minha irmã, Larissa. Essas 3 pessoas em especial sempre estiveram comigo desde que me entendo por gente e nunca me deixaram faltar nada. Sempre me deram atenção e carinho, sempre fazendo eu me sentir especial e querido.

Dedico também à minha namorada, Daniella Angelos, que foi a pessoa que mais esteve comigo durante todo o processo de realização deste trabalho. Foi uma grande companhia em todos os momentos e, com certeza, foi parte fundamental em me ajudar a ter resiliência para seguir em frente.

Agradecimentos

Raphael Rodrigues Pereira. Remerceio o meu orientador, Jacir Luiz Bordim, pela compreensão e paciência em todos os momentos, instruindo-me sempre a fazer o melhor e o mais profissional. Ajudando-me também em momentos difíceis, como na perdas de parentes. Estas atitudes deram-me a segurança necessária para seguir adiante.

Agradeço aos bons professores que tive a honra de conhecer durante a graduação, como Germana Menezes, Vander Ramos, Fernando Albuquerque, Pedro Rezende, Marcelo Mandelli, Flávio Leonardo, Marcelo Ladeira e Marcos Caetano, que auxiliara meu processo de formação.

Gratifico os funcionários da UnB pela gentileza e cordialidade, Sebastião Fonseca, Manoel Domingos e Tarcísio Ferreira, os quais foram meus companheiros durante várias noites na UnB. À Adriana Silva e ao Hebert Lima que também me fizeram companhia em vários dias da minha graduação durante anos. E ao Edson Carlos, da Secretaria de Administração Acadêmica, responsável pelo curso de Ciência da Computação, que ajudou-me a resolver vários problemas durante toda a graduação. Muito obrigado!

E, por fim, agradeço aos meus amigos, Jean Pierre, parceiro em vários trabalhos e disciplinas, juntamente com Michel Melo, Alexandro Gomes, Paulo Passos e Wilson Miranda, que lutaram comigo nessa árdua batalha por vários anos, virando noites na UnB, fazendo trabalhos intermináveis, estudando para provas dificílimas do Lucero. Agradeço ao Diego Marques, que me emprestou o livro do Sipser e me incentivou a não desistir, aconselhando-me com sabedoria. Ao Samuel Pala que esteve comigo em muitos momentos, compartilhando sua alegria e seus sorrisos. Ao Pedro Henrique, meu companheiro do trabalho de graduação, grande auxílio no término com sucesso deste trabalho. E finalmente, mas não menos importante, agradeço ao Paulo Ricardo e ao Fabrício Santana, que estivem comigo em vários bons e maus momentos de minha vida, especialmente desde a minha infância. Serão muitas lembranças boas de todos esses períodos.

Pedro Henrique Lima Ferreira. Gostaria de agradecer primeiramente ao meu orientador, Jacir Luiz Bordim, que teve muita paciência e compreensão durante todo este processo.

Também gostaria de agradecer a todas as pessoas com as quais tive contato através da Maratona de Programação na UnB. Participar desta comunidade e ajudá-la a crescer foi algo sensacional e mal consigo descrever a alegria que tive por fazer parte de tudo isso. Agradeço ao Matheus Pimenta por ter me apresentado a este mundo da progamação competitiva e por ter sempre acreditado em meu potencial na Maratona de Programação e também fora dela. Também gostaria de agradecer ao meu amigo José Leite que sempre foi um grande parceiro nesta área, sempre me desafiando e incentivando a evoluir. Agradeço ao professor Vinícius Borges, um dos professores mais humildes e respeitosos que conheci, e que sempre busca ajudar ao máximo os alunos. Agradeço muito também ao grande professor Edson Alves, professor da FGA, que é uma pessoa sensacional, muito humilde e que teve um papel fundamental no crescimento da Maratona de Programação no DF.

Por fim, agradeço ao meu companheiro de trabalho e amigo, Raphael, que teve bastante paciência durante todo este processo e foi peça fundamental para a realização deste trabalho.

Resumo

Com a grande popularidade do protocolo de Internet, usado para definir os endereços de rede para dispositivos conectados à Internet, criou-se um grande problema de esgotamento de endereços. Isto resultou na criação da Tradução de Endereço de Rede (do inglês, *Network Address Translation* - NAT), utilizado com o objetivo de compartilhar o uso de um único endereço de rede entre vários dispositivos. Seu amplo uso, acarretou em problemas relacionados a conectividade entre aparelhos em diferentes redes. No trabalho será realizada uma apresentação dos principais métodos utilizados para a travessia de NAT, destacando o Protocolo de Controle de Porta (do inglês, *Port Control Protocol* - PCP). Este protocolo, proposto pelo *Internet Engineering Task Force* e descrito na RFC 6887, permite que um dispositivo em uma rede privada faça um mapeamento de seu endereço de rede para um endereço externo e roteável, permitindo assim ser alcançável por diferentes nós na Internet. O objetivo é analisar as principais funcionalidades deste protocolo na prática, buscando integrar uma implementação deste com duas ferramentas de *backup*, o Bacula e o Rsync, para expor como esta solução de travessia de NAT pode ser utilizada em um cenário real com o objetivo de aumentar o escopo de uso destas ferramentas. Foi feita uma análise de performance dessas ferramentas em um cenário onde o uso do PCP é necessário, indicando o PCP como uma solução válida e eficiente para a realização de travessia de NAT.

Palavras-chave: PCP, protocolo, travessia NAT, *backup*.

Abstract

With the widespread popularity of the Internet protocol, which is used to distribute network addresses to devices connected to the Internet, a major problem of address exhaustion has been created. This resulted in the creation of the Network Address Translator, which is mainly used to share the use of a single network address across multiple devices. Its widespread use has led to problems related to connectivity between devices on different networks. The main methods used for NAT traversal were presented, with a special focus on the Port Control Protocol, PCP. This protocol, proposed by the IETF and described in RFC 6887, allows a host on a private network to map its IP address and port to a routable address, thus allowing it to be reachable by different nodes on the Internet. The goal is to analyze this protocol's main features in practice, integrating an implementation with two backup tools, Bacula and Rsync, to expose how PCP can be used in a real scenario in order to increase the usability of these tools. An analysis of performance of these tools was performed in a scenario where PCP is necessary, showing PCP as a valid and efficient solution for NAT traversal.

Keywords: PCP, protocol, NAT traversal, backup.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Abreviaturas e Siglas

ACK Acknowledgement Packet.

ADF Address-Dependent Filtering.

ADM Address-Dependent Mapping.

APDF Address and Port-Dependent Filtering.

APDM Address and Port-Dependent Mapping.

API Application Programming Interface.

ARPANET Advanced Research Projects Agency Network.

CD Compact Disk.

CGN Carrier-Grade NAT.

CIC Departamento de Ciência da Computação.

CIDR Classless Inter-Domain Routing.

CRLF Carriage Return and Line Feed.

DARPA Defense Advanced Research Projects Agency.

DHCP Dynamic Host Configuration Protocol.

DMZ Demilitarized Zone.

DNS Domain Name System.

DVD Digital Video Disc.

EDF Endpoint-Dependent Filtering.

EDM Endpoint-Dependent Mapping.

EIF Endpoint-Independent Filtering.

EIM Endpoint-Independent Mapping.

ENIAC Electronic Numerical Integrator and Computer.

FD File Deamon.

FTP File Transfer Protocol.

GB Gigabyte.

IANA Internet Assigned Numbers Authority.

IBM International Business Machines.

IGD Internet Gateway Device.

IP Internet Protocol.

ISP Internet Service Provider.

LSM Linux Security Modules.

MB Megabyte.

NAPT Network Address Port Translation.

NAT Network Address Translation.

NAT-PMP NAT Port Mapping Protocol.

OSI Open System Interconnection.

P2P Peer-to-peer.

PCP Port Control Protocol.

RFC Request for Comments.

SCP Secure Copy Protocol.

SIP Session Initiation Protocol.

SSH Secure Shell.

STUN Session Traversal Utilities for NAT.

SYN Synchronization Packet.

TCP Transmission Control Protocol.

TLS Transport Layer Security.

TURN Traversal Using Relay NAT.

UCLA University of California, Los Angeles.

UDP User Datagram Protocol.

UnB Universidade de Brasília.

UPnP Universal Plug and Play.

USB Universal Serial Bus.

VoIP Voice over Internet Protocol.

Capítulo 1

Introdução

Ao longo da história da humanidade surgiram vários avanços tecnológicos, no entanto nos tempos atuais as inovações têm ganhado uma velocidade inédita. Desde a descoberta do fogo e a sua respectiva manipulação pelo homem, assim como a partir dos primeiros equipamentos bélicos destinados à caça; os primeiros artefatos domésticos em 600.000 a.C. [8], e a invenção da roda, em aproximadamente 5000 a.C pelos sumérios [9], há uma crescente busca por inovações que facilitem a vida humana. A partir do século XX, a tecnologia proporcionou um salto nas inovações cujos marcos são a invenção da máquina de Turing, apresentada em 1936 por Alan Turing e o primeiro computador, denominado Computador Integrador Númerico Eletrônico (do inglês, *Electronic Numerical Integrator and Computer* - ENIAC), em [10]. A partir daí as transformações têm evoluído rapidamente e transformado paradigmas, especialmente com a invenção da Internet, em função da criação do protocolo TCP/IP em 1983 [10], até chegarmos ao cenário atual, com uma grande variedade de computadores e *smartphones* conectados entre si, com alto poder de processamento.

Com toda essa busca por novas tecnologias, a geração de dados cresceu exponencialmente, e, com isso, necessitou-se de um meio para armazenar os dados gerados. Inicialmente surgiram os cartões perfurados, em 08 de janeiro de 1889, por Herman Hollerith [11], contendo 80 *bytes* de dados; a seguir foram criadas as fitas magnéticas em 1950, utilizadas pela primeira vez pela *International Business Machines* (IBM), as quais armanezavam um total de 0,76MB de dados. A partir de 1956 os discos rígidos começaram a ser usados, introduzidos novamente pela IBM, arquivando, dessa vez, uma quantidade de 4,4MB. Já em 1969, os disquetes emergiram com grande popularidade por serem menores e por possuírem uma boa capacidade de armazenamento na época. Suas primeiras versões guardavam 80kB de informações, mas, em 1990, já chegavam a 250MB cada. Em 1979 os discos compactos foram criados pela Sony e pela Philips, podendo estocar 700MB de dados, correspondendo a cerca de 497 disquetes 3 1/2". Com isso, a popularidade do

Disco Compacto (do inglês, *Compact Disk* - CD) foi enorme, pois disponibilizava uma melhor portabilidade com um ganho considerável de espaço, tornando os disquetes uma segunda opção de *backup* a partir de 1982 [12]. Em 1997 o primeiro Disco Digital Versátil (do inglês, *Digital Video Disc* - DVD) foi lançado pela Sony, Pioneer e Toshiba, podendo armazenar 8,5GB de dados [13], tendo um aumento significativo em comparação ao seu precursor, o CD. Após, em 1998, surgiram então os dispositivos *flash* portáteis, podendo ser ligados na entrada Porta Universal (do inglês, *Universal Serial Bus* - USB) do computador. A partir desse momento, a demanda por armazenamento cresceu em ritmo acelerado, sendo necessário o desenvolvimento de um novo meio de armazenar dados que possibilitasse acesso rápido. Surgiu então o conceito de nuvem, uma vez que, de acordo com pesquisa realizada pela Seagate, em 2019, a quantidade de dados gerados ultrapassará os 33 *zettabytes* [14].

Tendo em vista todo o avanço na área de computação ao longo dos anos, a Internet desempenhou papel fundamental, crescendo velozmente. Desde 1969, com a criação da primeira rede entre a universidade de Stanford e a Universidade da Califórnia em Los Angeles(do inglês, *University of California, Los Angeles* - UCLA) [15], até hoje, a Internet obteve alcance global, contribuindo para o processo de globalização em curso na sociedade moderna. Assim, a Internet tornou-se a principal responsável pelo avanço do conhecimento, uma vez que proporcionou sua difusão, permeando quase tudo o que se conhece hoje em tecnologia, por conta da união de artefatos e conhecimentos que já existiam. Um exemplo elucidador deste fato é a transformação ocorrida quanto aos meios de comunicação; como a comunicação por voz que utilizava uma central de telefonia e que por meio da Internet passou a ser executada de forma automatizada com o WhatsApp, o Skype, o Telegram e vários outros *softwares*. Esta mudança foi possível devido à intersecção de dados que passaram a existir após a sua criação, como, por exemplo, as chamadas de vídeo *online*. Destarte, notou-se a grande quantidade de dados sensíveis e descartáveis que seus usuários começaram a produzir, podendo ser observados através da quantidade de sites que foram sendo criados ao longo dos anos, como mostra a Figura 1.1, crescendo ainda mais a premência de sistemas de armazenamentos seguros.

Sendo assim, surgiu a necessidade da criação de um sistema que possibilitasse o armazenamento desses dados, de modo que caso acontecesse algo inesperado, eles pudessem ser recuperados facilmente. O termo *backup* (ou cópia de segurança) foi posto em prática em 1951, justamente com esse propósito, para o armazenamento de dados com o objetivo de obter maior segurança contra erros, para isso utilizavam-se cartões perfurados para armazenamento [16].

Com toda a popularização da Internet desde a sua criação e o com o aumento da quantidade de dispositivos que passaram a utilizá-la, foi necessária a criação de uma

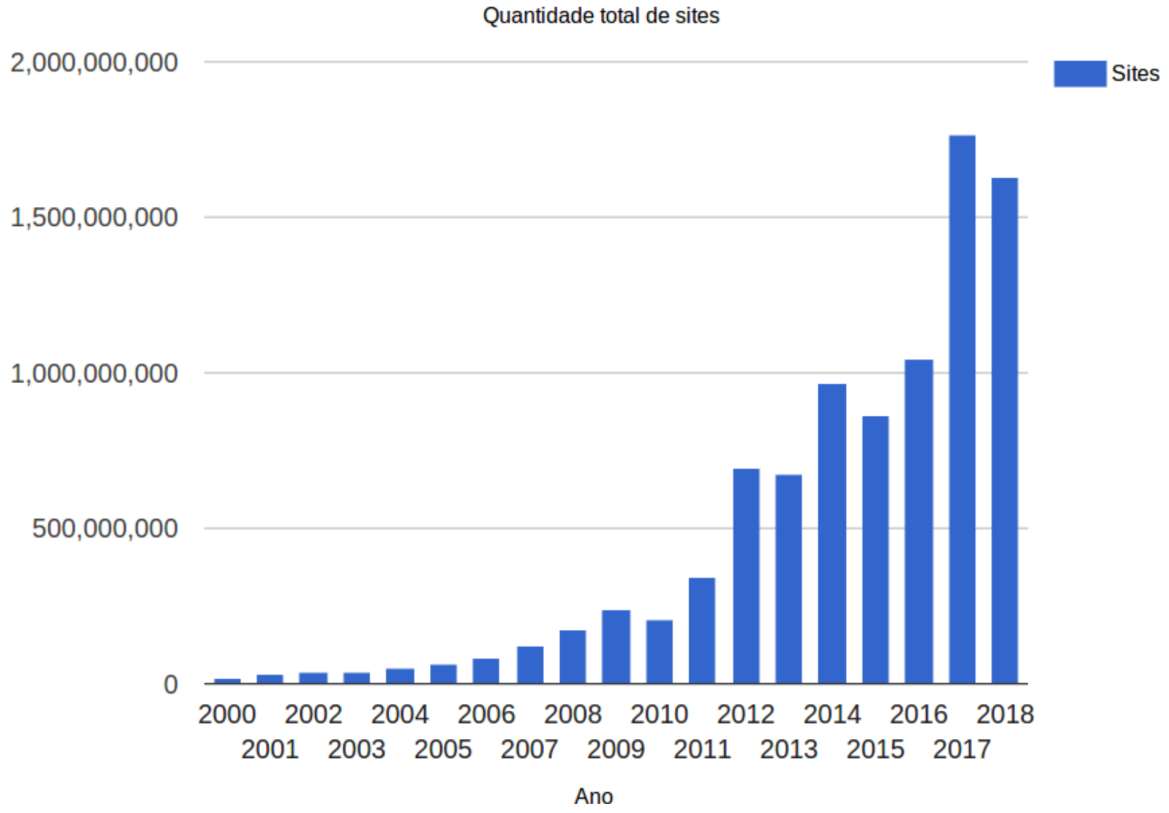

Figura 1.1: Crescimento da quantidade de *websites* de 2000 a 2018 [1].

ferramenta que pudesse fazer com que essas conexões fossem disponibilizadas para todos os dispositivos. Pois, a medida que cada dispositivo novo conectasse à Internet, cada endereço de Protocolo de Internet (do inglês, *Internet Protocol* - IP) era reservado a ele [2]. Sendo assim, com o passar do tempo, começou a ocorrer uma escassez da quantidade de IPs públicos utilizados, sendo necessária a implementação de uma nova tecnologia que fosse capaz de aumentar essa capacidade de IPs ou que possibilitasse uma maior quantidade de dispositivos em um mesmo Protocolo de Internet público [17]. Desta forma surgiu o que se conhece hoje como os NATs, que possibilitaram que vários dispositivos em uma rede privada conectassem a somente um endereço externo, possibilitando, desta forma, que uma maior quantidade de *hosts* pudessem utilizar a Internet[18].

Visto que o NAT foi uma ótima solução para a escassez de IPv4 restantes no mundo, houveram várias dificuldades em relação ao seu uso, como por exemplo um meio de realizar uma conexão de um servidor real para um cliente real atrás de um NAT. Com isso surgiram várias restrições quanto ao seu uso, tendo que serem criados os métodos de travessia de NAT, que serão elencados no Capítulo 3. Com esses métodos foram possíveis os acessos a dispositivos que se encontram em IPs falsos atrás do NAT, possibilitando, deste modo, uma variedade de benefícios e possibilidades com o seu uso, desde uma chamada de vídeo,

até ao jogar-se em rede.

1.1 Problema

Com o surgimento e a popularização do NAT, por implementar uma solução fácil e barata [19] para o problema da disponibilidade de endereços IPv4 no mundo, deparou-se com problemas como, por exemplo, a realização de uma conexão par a par. Aliado a isso, foi suscitado também um problema atual em um *software* de *backup*, denominado Bacula. Em que, atualmente, nele somente é possível realizar pedidos de *backups* através de um cliente para um servidor na rede externa, não sendo possível o contrário. Desta forma, foi pensado em realizar-se a incrementação do protocolo de controle de portas para que essa funcionalidade fosse realizada.

1.2 Objetivos

Face ao preâmbulo já elencado, este trabalho tem por objetivo realizar uma avaliação do uso do PCP, de travessia de NAT, na tentativa de acrescentar uma nova funcionalidade ao Bacula, um *software* gratuito de *backup*, como será visto no capítulo 5. Funcionalidade esta que permite que um servidor possa se conectar em um cliente, que se encontrará atrás de uma rede com NAT, para a realização de um *backup*.

1.3 Metodologia

Para a validação do PCP foi desenvolvido um sistema capaz de realizar uma comunicação de um computador que agiria como um cliente PCP atrás de um NAT, um segundo computador que agiria como um servidor PCP, que pudesse realizar operações de MAP e PEER, que serão explicados na seção 3.2 deste trabalho. Por fim, um terceiro computador foi utilizado como roteador NAT.

Já para a utilização do PCP como uma solução válida de travessia de NAT em *backups*, foram montados dois cenários de teste. No primeiro foi criado um ambiente contendo dois computadores na mesma rede, sendo um computador possuindo o cliente do PCP e um outro computador abarcando o servidor, rodando dentro de uma máquina virtual. Neste cenário foram realizados dois testes, um deles sendo o cliente e o servidor utilizando o *software* Bacula e o segundo teste, com o cliente e o servidor utilizando o Rsync. No segundo cenário, o cliente se situa atrás do NAT e o servidor fora dele, como será mostrado posteriormente no Capítulo 5.

1.4 Organização do trabalho

O trabalho será organizado em sete capítulos da seguinte forma:

- Capítulo 2: Introduzidos conceitos e protocolos considerados indispensáveis para o entendimento do trabalho. Serão apresentados desde como a Internet começou a ser pensada e desenvolvida, a partir do uso dos modelos de padronização, até a descrição da ferramenta que será utilizada para a verificação de campos de mensagens. Passando pela definição e explicação dos protocolos IP e DHCP, como também pela travessia de NATs, dizendo o seu funcionamento e suas especificidades.
- Capítulo 3: Descrição dos protocolos utilizados hoje como soluções de travessia de NAT, bem como a apresentação de outras propostas de travessias de NAT, explicando porque foi utilizado o protocolo PCP como uso para a realização deste trabalho.
- Capítulo 4: Serão realizados casos de testes para que seja possível validar a utilização do protocolo de controle de portas, para ser utilizado no próximo capítulo deste trabalho. Estes testes serão úteis para verificar o funcionamento deste protocolo, para averiguar se ele funciona corretamente como descrito em sua RFC.
- Capítulo 5: Serão efetuados os devidos testes, para a verificação da possibilidade do uso do mesmo em *softwares* de *backup*. Para isto, será verificado também o uso de um segundo programa de *backup* gratuito, para que seja possível realizar uma comparação com a aplicação escolhida para a validação do protocolo neste trabalho.
- Capítulo 6: Será apresentado qual foi o resultado obtido com o decorrer dos testes e dos conceitos utilizados no trabalho, visando o protocolo PCP como uma solução válida para a utilização da aplicabilidade de travessia de NAT em *backups*.
- Capítulo 7: Aqui, apenas foi idealizado e incrementado no trabalho para os conceitos que demandam um pouco mais de conhecimentos, para leitores que não sejam da área de pesquisa, possam acessá-los mais facilmente.

Capítulo 2

Fundamentação Teórica

Este capítulo apresenta conceitos básicos de rede, fundamentais para o entendimento do trabalho. A começar pelo modelo de Interconexão de Sistemas Abertos (do inglês, *Open System Interconnection* - OSI), que tem como objetivo padronizar todo o sistema de comunicação de máquinas na Internet. Este modelo, como veremos em 2.1.2, é originalmente dividido em 7 camadas. Dentre elas, há um foco mais específico na camada de rede, responsável por acomodar o IP. Em 2.2, há revisão das principais vantagens e desvantagens deste protocolo, em especial sua quarta e mais popular versão, o IPv4.

Em 2.4, é visto o *Network Address Translator* (NAT), ou tradutor de endereço de rede, com suas diferentes variações, proposto para solucionar um dos principais problemas criados com a grande popularidade do IPv4. Nesta mesma seção, encontra-se a explicação do problema de travessia em NAT e uma breve descrição de alguns métodos para resolvê-lo. Por fim, em 2.5, há a explicação de *sniffers* de rede e uma apresentação, em específico, do Wireshark, que é a ferramenta de captura que utilizaremos neste trabalho.

2.1 Pilha de protocolos da Internet

Com o avançar da Internet e das redes, as empresas estavam investindo cada vez mais em tecnologia para que pudessem ter alguma vantagem competitiva em relação a suas concorrentes. Porém, caso um usuário quisesse se conectar a outro usuário com equipamento de um fabricante diferente, ele não conseguiria, pois cada fabricante de computadores criava sua própria topologia, seus próprios protocolos e suas próprias tecnologias. Sendo assim, a comunicação entre dois dispositivos de fabricantes diferentes era inviável, uma vez que apresentava diversos problemas de interoperabilidade entre os dispositivos [20].

Ao deparar-se com tal situação, as empresas decidiram entrar em consenso para a criação de um modelo padrão de rede para que todos pudessem utilizar o mesmo protocolo de serviço e, assim, possibilitar a conexão entre equipamentos de fabricantes diferentes.

Dessa forma, foi possível resolver esse problema através dos dois modelos que serão apresentados a seguir, o modelo referência de OSI e o modelo TCP/IP.

2.1.1 Modelo TCP/IP

Em 1969, o modelo TCP/IP foi desenvolvido pela Agência de Projetos de Pesquisa Avançada de Defesa (do inglês, *Defense Advanced Research Projects Agency* - DARPA) visando uma solução para a necessidade de comunicação entre diversos computadores e organizações militares, denominando-se como um projeto de Rede da Agência para Projetos de Pesquisa Avançada (do inglês, *Advanced Research Projects Agency Network* - ARPANET), posteriormente chamado de modelo de referência TCP/IP [2].

O objetivo dele era fazer a disponibilização de enlaces de comunicação de alta velocidade utilizando comunicação em rede através de comutação de pacotes, visando uma rede robusta ante a um possível ataque de inimigos [2]. Sendo assim, o protocolo era capaz de encontrar e identificar uma melhor rota possível entre dois nós, procurando também rotas alternativas para o deslocamento até o nó destino, caso um nó fosse derrubado ou deixasse de operar repentinamente. Destarte, era necessária uma arquitetura flexível, capaz de se adaptar a aplicações com requisitos distintos, como transferência de arquivos e transmissão de dados de voz em tempo real [2].

Na descrição das camadas abaixo foram utilizadas como referências majoritárias o texto contido na tradução do livro do Tanenbaum e Andrew, Redes de computadores (2003) [2].

- Camada de enlace

  Essa camada tem como funcionalidade a compatibilidade com a tecnologia da rede com o IP. Isto posto, ela recebe os datagramas IP da camada de Internet e os transmite através de uma rede específica, passando por uma série de roteadores. Para isso, os endereços IP, endereços lógicos, são traduzidos para os endereços físicos dos *gateways* conectados a essa rede [21]. Dois exemplos de protocolos dessa camada são os protocolos Ethernet e Wi-Fi [4].

- Camada de Internet (também chamada de camada de rede)

  Devido à necessidade de uma rede mais robusta e volátil, verificou-se a necessidade de uma rede de comutação de pacotes em uma camada de interligação de redes sem conexões [17], sendo caracterizada por: não possuir reserva de recursos; o meio ser compartilhado; o encaminhamento de pacotes ser feito nó a nó; e os pacotes possuírem endereço de destino, podendo, no entanto, não ser entregues devidamente. Em vista disso, segundo Soares *et al.* [21], essa camada é responsável pela transferência

de dados através da Internet, desde a máquina de origem até a máquina de destino. Ela recebe pedidos da camada de transporte para a transmissão de pacotes que, ao solicitar a transmissão, informa o endereço do dispositivo onde o pacote deverá ser entregue. Desta forma, o pacote é encapsulado em um datagrama IP, executando também o algoritmo de roteamento, determinando se esse pacote pode ser entregue diretamente ao destinatário ou se deve ser transmitido a um *gateway*. Dessarte, o datagrama é passado para a interface de rede para então ocorrer a transmissão. Um exemplo de protocolo dessa camada é o protocolo IP [4], que será visto com mais detalhes em 2.2.

- Camada de transporte

  Ela permite uma comunicação fim a fim entre as aplicações, como acontece também na camada de transporte do modelo de referência OSI. Nessa camada são definidos dois protocolos fim a fim, o Protocolo de Contole de Transmissão (do inglês, *Transmission Control Protocol* - TCP) e o Protocolo de Datagrama de Usuário (do inglês, *User Datagram Protocol* - UDP). Sendo o primeiro deles um protocolo orientado a conexões confiáveis, que faz controle de erros e permite a entrega sem falhas de um determinado dispositivo para qualquer computador da Internet, realizando também um controle de fluxo, impedindo sobrecarga do receptor. No IPv4, ele faz uma fragmentação dos *bytes* de entrada em mensagens discretas, passando cada uma das partes para a camada de Internet. Ao recebê-los na máquina alvo, o processo TCP volta a montar as mensagens recebidas. O segundo protocolo é um protocolo não confiável e sem conexão, requerido para aplicações que não demandam controle de fluxo e nem a manutenção em sequência das mensagens que foram enviadas, desenvolvendo seu próprio recurso para resolvê-lo [2].

- Camada de aplicação

  Camada que contém todos os protocolos de nível mais alto, tais como o Protocolo de Transferência de Arquivos (do inglês, *File Transfer Protocol* - FTP), o Protocolo de Transferência de Correio Simples (do inglês, *Simple Mail Transfer Protocol* - SMTP), o protocolo de Sistema de Nomes de Domínios (do inglês, *Domain Name System* - DNS), o Protocolo de Transferência de Hipertexto (do inglês, *Hypertext Transfer Protocol* - HTTP), entre vários outros. Para isso, os usuários utilizam programas para acessar esses serviços, utilizando a camada de transporte para enviar e receber dados. Neste trabalho será utilizado com mais frequência o Protocolo de Configuração Dinâmica de *Host* (do inglês, *Dynamic Host Configuration Protocol* - DHCP) (que utiliza o protocolo UDP disposto na camada de transporte), e para seu melhor entendimento ele será descrito a seguir.

Há também a pilha de protocolos da Internet, que é o utilizado atualmente. Ela se assemelha às camadas do modelo TCP/IP, distrinchada em 5 camadas. Cada uma das camadas citadas anteriormente possui o mesmo conceito e as mesmas funcionalidades, diferentemente da camada física, que não existe no modelo TCP/IP. Ela é responsável pelo envio dos *bits* individuais que estão dentro dos pacotes da camada de enlace de um nó atual para o próximo [4].

2.1.2 Modelo OSI

Visando acabar com o bloqueio de comunicação entre redes de diferentes propriedades, foi criado o modelo OSI, em 1984 [22], para tratar da interconexão de sistemas abertos.

Apesar de primordial e de apresentar técnicas utilizadas até hoje, ele não é amplamente implementado, pois ele, segundo Tanenbaum (2003), não é uma arquitetura de rede, por não especificar exatamente os protocolos e os serviços que devem ser usados em cada camada. O modelo apenas aborda o que cada camada deve fazer [2] e não possui flexibilidade entre as camadas.

Este modelo é separado em sete camadas que informam como os dados devem ser tratados, desde os bits de um canal de comunicação até a aplicação que o usuário está executando, como mostra a Figura 2.1. Segundo Kurose [4], as camadas de aplicação, transporte, rede, enlace e a camada física da pilha de protocolos da Internet detalhadas no modelo TCP/IP possuem as mesmas definições do modelo OSI, como foram apresentadas na Seção 2.1.1. Sendo assim, serão detalhadas somente nesta seção somente as camadas de apresentação e de sessão.

- Camada de sessão

  Ela permite que usuários de dispositivos distintos estabeleçam sessões entre eles, cuidando dos erros e fazendo uma administração dos registros das transmissões. Essa camada é vista como responsável por fazer o controle do fluxo de dados entre os nós [23].

- Camada de apresentação

  Na camada de apresentação há uma preocupação maior de como as informações são transmitidas, em termos sintáticos e semânticos, gerenciando estruturas de dados abstratas e permitindo a troca para estruturas de dados a níveis mais altos [2]. Ou seja, ela disponibiliza serviços que permitem que as aplicações interpretem os significados dos dados que serão trocados, como por exemplo a compressão e a codificação dos dados e a descrição de dados [4].

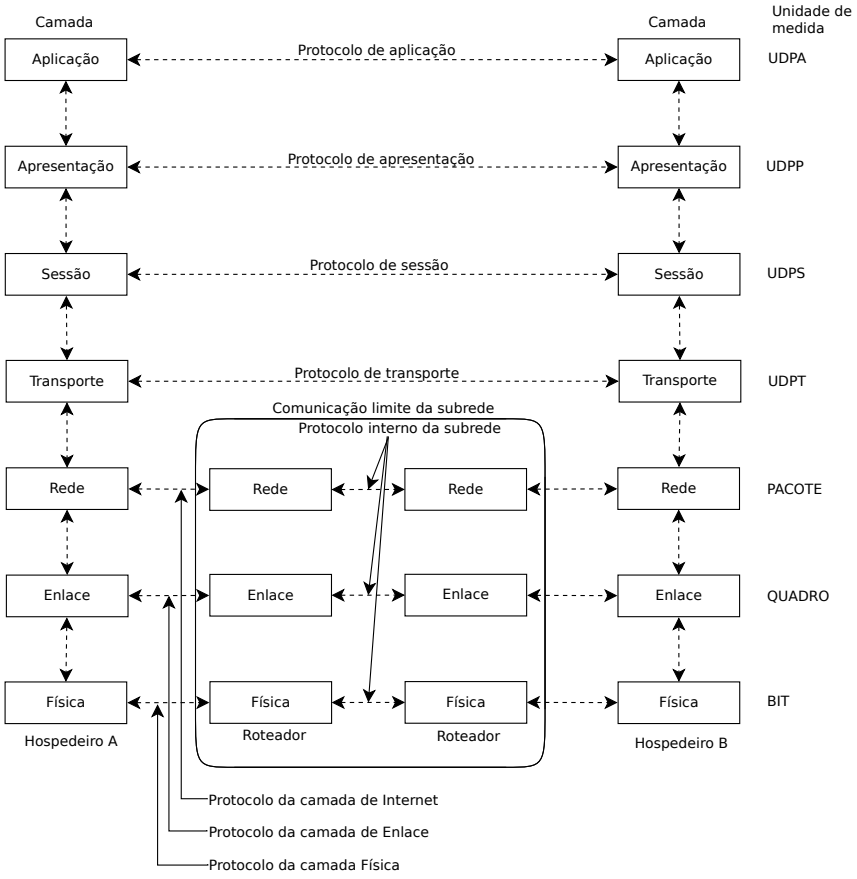

Figura 2.1: Modelo de referência OSI [2].

2.2 IP

Para ter um aprofundamento maior da camada de rede, é necessário entender o protocolo IP. De acordo com [2], um datagrama IP consiste em uma parte de cabeçalho, com 20 *bytes* fixos e uma parte opcional de texto de tamanho variável. Os principais campos, essenciais para o entendimento do trabalho, são os de endereço de origem e de endereço de destino. A princípio, cada *host* e roteador na rede tem um endereço único e exclusivo de IP que codifica seu número de rede e número de *host*. Um endereço de IP corresponde somente a uma interface de rede, logo é possível que um *host* tenha dois endereços de IP ao mesmo tempo, caso esteja conectado em duas redes diferentes através de duas interfaces diferentes.

Em geral, os endereços de rede do IPv4, que são números de 32 *bits*, são escritos em notação decimal com pontos, onde cada ponto separa 1 *byte* do número. Logo, o valor 0.0.0.0 é o menor número de endereço de IP enquanto o 255.255.255.255 é o maior. Como dito anteriormente, esse endereço codifica o número da rede e número do *host*, de modo que os *bits* mais significativos indicam qual o número da rede enquanto o resto indica o

número do *host* naquela rede. Por muitos anos, os endereços de IP foram divididos em 5 categorias, listadas na Figura 2.2 [4].

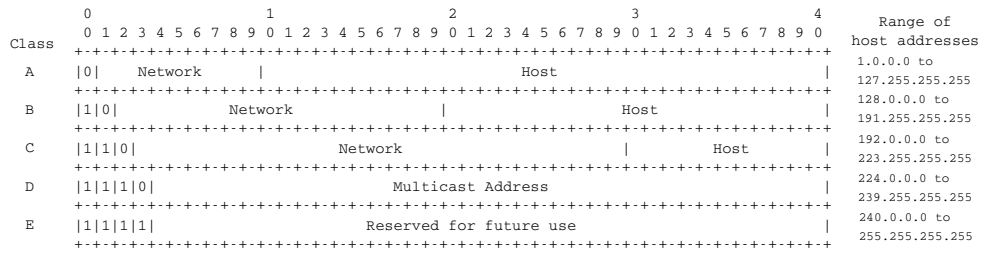

Figura 2.2: Classes de formatos de endereços de IP [3].

A classe de formato A suporta 128 redes com aproximadamente 16 milhões de hospedeiros cada. A classe B permite aproximadamente 16 mil redes com até 64 mil *hosts* em cada. A classe C possibilita 2 milhões de redes com até 256 *hosts* cada. Os endereços de multidifusão da classe D são utilizados para fazer a transmissão de um datagrama para vários *hosts*, e, por fim, ainda existem endereços de IP da classe E reservados para o futuro. Segundo Kurose [4], essa divisão dos endereços em classes foi abandonada e, posteriormente, as divisões entre números de rede e *host* tornaram-se mais flexíveis, possibilitando um usuário a escolher quantos *bits* ele deseja para numerar sua rede. Para esta nova divisão, deu-se o nome de *Classless Inter-Domain Routing*, conhecido como CIDR. Nele, a rede pode ter tamanho variável, usa-se a seguinte notação padrão para indicar quantos bits representam o número da rede:

- 192.168.0.0/24 indica um endereço de IP onde os primeiros 24 bits são fixos representando o número da rede e os 8 bits restantes representam o número do *host*. Logo, há 256 endereços IPv4 nesta rede, de 192.168.0.0 até 192.168.0.255 inclusive, com 192.168.0.255 sendo o endereço de broadcast para a rede [2].

Para finalizar esta seção, é necessário explicar o que é uma rede privada, um conceito importante para o entendimento do trabalho. A RFC 1918 [24] propôs reservar três faixas de endereçamento IP somente para redes privadas. As faixas são de 10.0.0.0 à 10.255.255.255, 172.16.0.0 à 172.31.255.255 e 192.168.0.0 à 192.168.255.255. Isto significa que nenhum dispositivo conectado à Internet pode ter estes endereços de IP. Além disso, estes endereços são denominados, aqui neste trabalho, de falsos ou não roteáveis, que são números de IPs de *hosts* em redes privadas. Vale ressaltar também os endereços de *loopback* que, de acordo com a RFC 5735[25], estão na faixa de IP 127.0.0.0 à 127.255.255.255 (ou 127.0.0.0/8 na notação CIDR) e são reservados para a comunicação com o computador local. Quaisquer pacotes enviados para estes endereços ficarão no computador que os gerou e serão tratados como se fossem pacotes recebidos pela rede.

2.3 DHCP

De acordo com a RFC 2131 [26], o DHCP provê parâmetros de configuração para *hosts*, sendo composto por dois componentes, um protocolo para entregar parâmetros de configuração específicos para cada *host* de um servidor DHCP e um mecanismo para a alocação de endereços de rede para os hospedeiros. Ele é construído através de um modelo cliente-servidor, no qual o servidor aloca endereços de rede e entrega esses parâmetros para configurar dinamicamente os *hosts* (clientes). O texto referente a esta seção é baseado no texto contido na RFC 2131 [26].

Esse processo começa com o cliente e o servidor construindo mensagens DHCP, completando os campos fixos e acrescentando dados no campo de opções de tamanho variável. Após preencher todos os campos de mensagem, são configuradas as portas de recepção de requisições do servidor, sendo ela a porta 67, e a do cliente, porta 68, sempre utilizando UDP como protocolo de transporte [26]. Sendo assim, no caso de uma petição, por parte do cliente, para a alocação de um novo endereço de rede (ocorrendo a primeira conexão à rede), as mensagens são transmitidas como se seguem (omitindo passos específicos de erros):

1. Um cliente transmite uma mensagem DHCPDISCOVER em sua subrede física local.
2. Depois de receber a mensagem do cliente, o servidor aloca um novo endereço, e em seguida, ele checa se esse endereço já está sendo usado por outro cliente. Após a validação, ele pode responder ao cliente com uma mensagem de DHCPOFFER, que inclui o endereço da rede no campo yiaddr.
3. Sendo assim, o cliente recebe uma ou mais mensagens do tipo DHCPOFFER de um ou mais servidores, devendo escolher o que foi pedido pelos parâmetros de configuração na mensagem enviada pelo servidor. Sendo assim, o cliente envia agora uma mensagem DHCPREQUEST, incluindo o identificador do servidor, não mudando o valor de "secs" definido na mensagem de DHCPDISCOVER.
4. O servidor, por sua vez, recebe a mensagem e confirma a ligação com o cliente e faz um anexo ao *storage* do cliente, enviando, em seguida, uma mensagem de DHCPACK para o cliente.
5. Recebendo a mensagem de DHCPACK, por parte do cliente, com todas as configurações de parâmetros corretas, o cliente validará essas informações dos parâmetros e armazenará o tempo de concessão, contido na mensagem. Sendo assim, após a verificação, a configuração do cliente será realizada com sucesso.

6. Caso o cliente queira cancelar a concessão dele na tabela de DHCP do servidor, ele necessitará enviar uma mensagem de DHCPRELEASE pedindo para que esse procedimento seja realizado.

Essas mensagens podem ser visualizadas também através da Figura 2.3.

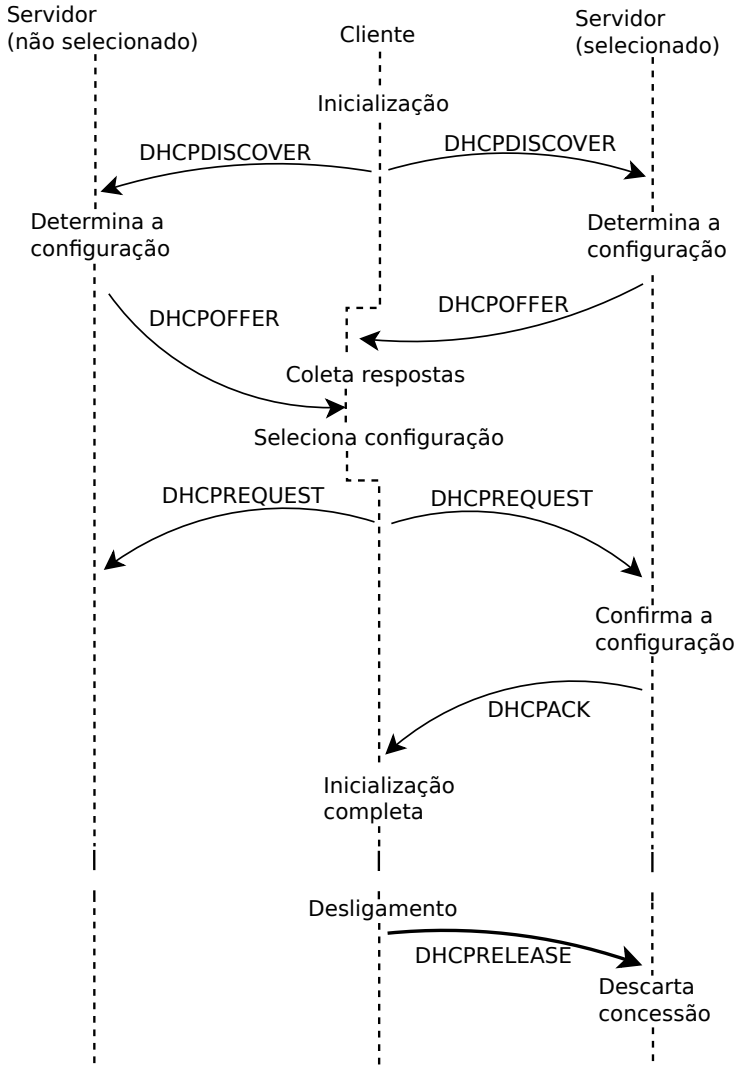

Figura 2.3: Troca de mensagens entre cliente e servidor DHCP na alocação de um novo endereço de rede.

2.4 Travessia NAT

Tendo em vista um aumento exponencial do número de aparelhos que necessitam de um IPv4 externo único, ao longo do crescimento da Internet, foi necessário criar um algoritmo para que fosse possível armazenar uma maior quantidade de IPs para uma única rede.

Sendo assim, pensou-se em criar o que conhecemos hoje como o NAT. Nele é possível fazer uma tradução de endereços falsos, visíveis somente para *hosts* que se encontram na mesma rede, para endereços reais, visíveis por qualquer *host* na Internet. Assim, ele funciona como uma caixa preta, que ao conectar-se um dispositivo no roteador, ele fará uma mudança de um IP externo para dispositivos conectados a IPs internos a uma rede. Um NAT nada mais é do que um método cujos endereços IPs são mapeados de um domínio a outro, na tentativa de prover um roteamento transparente aos *hosts* [18], como mostra a Figura 2.4. Para o restante da seção, os textos foram escritos baseados na RFC 2663 [18].

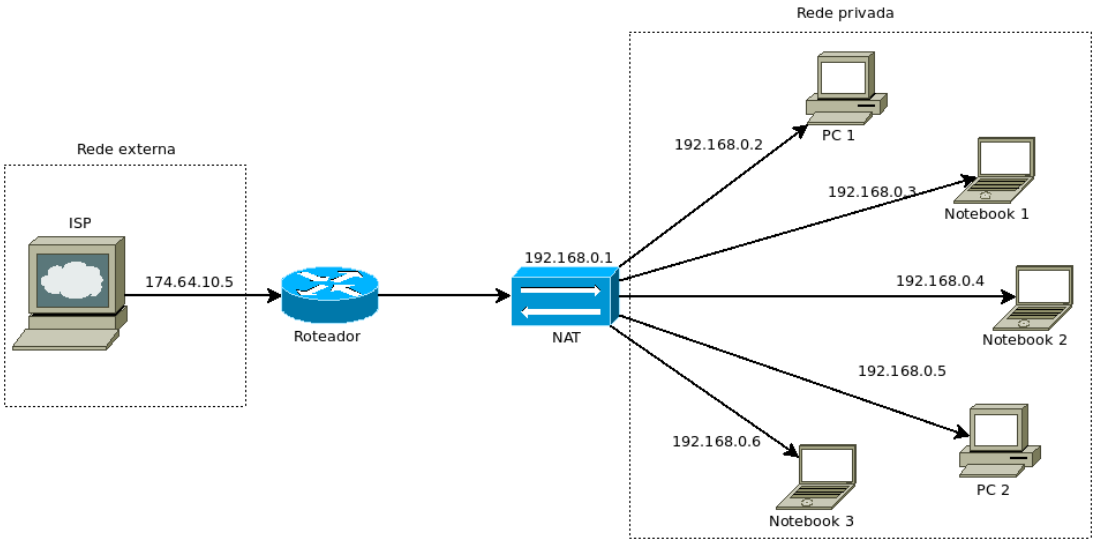

Figura 2.4: Funcionamento do NAT.

2.4.1 NAT tradicional

O NAT tradicional permite que os *hosts* dentro de uma rede privada tenham acesso transparente a *hosts* de uma rede externa, de modo que seus IPs sejam únicos, sendo assim, um nó na rede privada atrás do NAT pode iniciar sessões com nós fora do NAT, mas o contrário não é possível. Esse tipo se divide em duas categorias, no NAT básico e na Tradução de Endereço e Porta de Rede (do inglês, *Network Address and Port Translation* - NAPT). Na primeira categoria, um bloco de endereços externos é reservado para a conversão de endereços de hospedeiros em um domínio privado, a partir da necessidade da criação de sessões para o domínio externo. Ou seja, um conjunto de endereços é reservado para ser mapeado para *hosts* internos, a fim de permitir sessões de comunicação de entrada para eles. Já na segunda categoria, vários *hosts* atrás do NAT podem utilizar

o mesmo endereço externo, mas com portas diferentes, traduzindo apenas o endereço e as portas da origem e destino [18].

2.4.2 Bi-direcional NAT

Nesse tipo, as sessões podem ser iniciadas tanto a partir de um dispositivo na rede externa, quanto a partir de um dispositivo na rede interna, pois um nó atrás do NAT é mapeado unicamente para um endereço externo, tanto estaticamente como dinamicamente. Para que seja possível acessar um computador, por exemplo, atrás do NAT, é necessária a utilização de DNS para resolução de endereços [18].

2.4.3 *Twice* NAT

Diferentemente dos dois tipos anteriores, o *twice* NAT somente traduz um dos endereços de origem ou destino. É utilizado, principalmente, quando domínios privados e públicos possuem duplicatas. Como, por exemplo, caso uma rede utilize o espaço de endereçamento de IP de 100.100.100.0/24 e que contenha um site Y na rede externa que utilize o IP de 100.100.100.100. Se um computador X com o IP 100.100.100.4 tentar conectar no site citado, ele não o fará, pois ele tentará achar na rede interna o servidor 100.100.100.100, mas não será encontrado [18]. O fluxo do datagrama será do Computador X (Privado) para o Site Y (Público). Sendo assim, o *twice* NAT funcionará da seguinte maneira:

O Computador X envia uma requisição a um servidor DNS na rede pública para se conectar ao Site Y. O servidor, por sua vez, envia o IP do Site Y descoberto para o roteador da rede privada, como mostra a Figura 2.5. Posteriormente, o roteador encaminha a resposta ao Computador X, contendo o endereço de IP temporário traduzido (156.89.90.1) do Site Y, conforme consta na Figura 2.6.

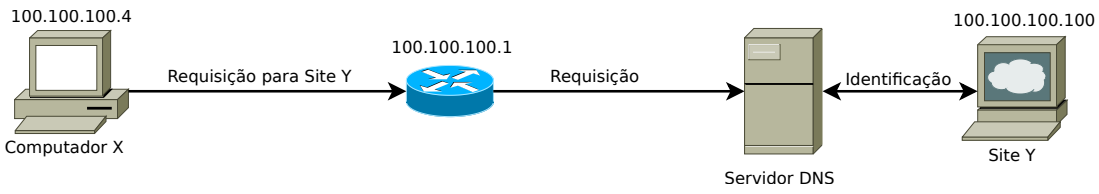

Figura 2.5: Requisição do Computador X.

Após a tradução do endereço, o Computador X pode se comunicar normalmente com o Site Y, de forma que ele somente saiba o endereço de IP falso disponibilizado pelo roteador de sua rede privada.

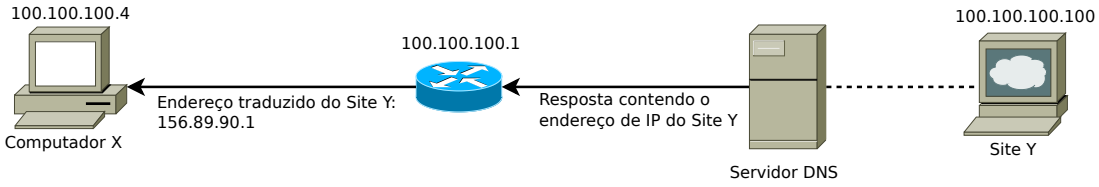

Figura 2.6: Resposta do servidor DNS.

2.4.4 *Multihomed* NAT

Nessa última configuração uma rede privada utiliza vários NATs com sessões ativas passando para um dispositivo NAT distinto, dependendo do destino almejado [27].

2.4.5 Métodos de travessia

O Protocolo de Controle de Portas, que será abordado nesse trabalho e explicitado no próximo capítulo, possui várias aplicabilidades, dentre elas o uso de *webcam* em uma rede privada; ligação por meio da Internet, utilizada em jogos, *backups*, dentre outras diversas aplicações [28]. Ele possui a intenção de criar mapeamentos de IPs externos para um IP e porta interna de uma rede para o uso dentro de aplicações. Sendo assim, o intuito desse trabalho foi realizar um mapeamento de computadores atrás de um NAT para a comunicação de um servidor em uma rede externa. Para isso, foi necessário aprender as técnicas de travessia NAT bem como os protocolos que a implementam, que serão apresentados no próximo capítulo. Sendo elas:

- Retransmissão

  De acordo com a RFC 5128 [29], esse método é o mais confiável, mas o menos eficiente para implementar a comunicação ponto a ponto na presença de um dispositivo NAT, fazendo com que uma comunicação Par a Par (do inglês, *Peer to Peer* - P2P) se pareça com uma rede de comunicação entre cliente e servidor. Nesse método, dois nós estão em uma rede externa ao NAT para retransmitir mensagens entre eles, como mostra a Figura 2.7. Esse método é utilizado no protocolo de Travessia Utilizando Retransmissão em torno do NAT (do inglês, *Traversal Using Relays around NAT* - TURN) [6], que será descrito na Seção 3.4.

- Conexão reversa

  Neste método, um dos nós precisa estar fora do NAT. Um nó X, dentro do NAT, registra seu endereço em um servidor de encontro. Como o nó Y, fora do NAT, não pode conectar diretamente no nó X, ele utiliza do servidor de encontro para retransmitir a requisição de conexão para o nó X, dizendo que deseja realizar uma

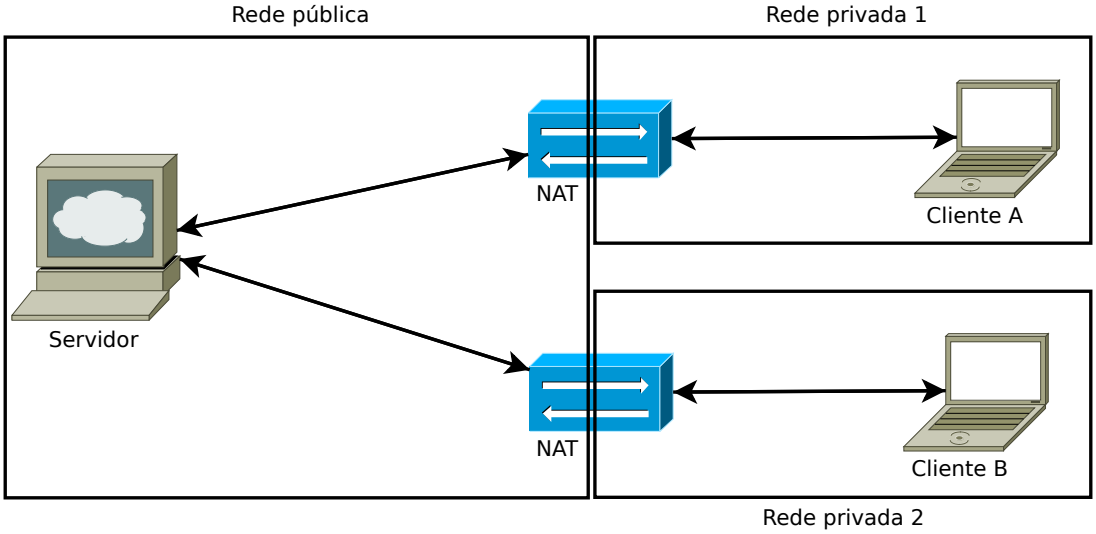

Figura 2.7: Travessia NAT utilizando retransmissão.

conexão reversa com ele. Uma vez recebida a mensagem, o cliente X, abre uma conexão TCP com o cliente Y, através do IP/Porta público de Y [30], como indica a Figura 2.8.

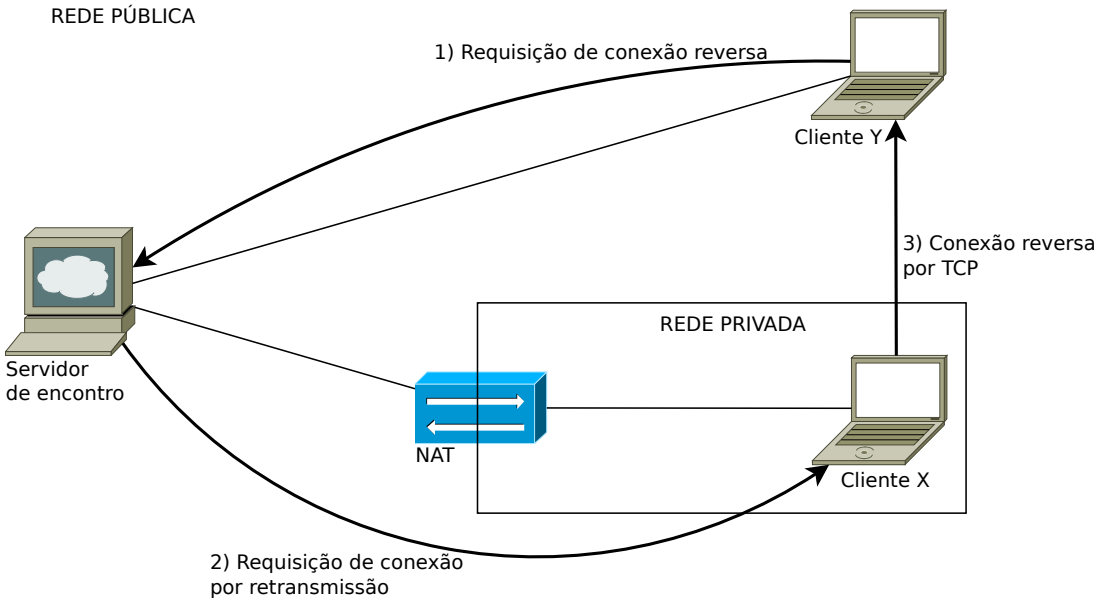

Figura 2.8: Travessia NAT utilizando conexão reversa.

- *Hole Punching*

  No método de *hole punching* há duas divisões, um primeiro método que utiliza conexões UDP e um segundo método que utiliza conexões TCP. A seguir os dois métodos serão explanados brevemente.

O primeiro método, possuindo dispositivos em NATs distintos, que é o cenário mais comum [30], permite que dois clientes X e Y façam uma comunicação P2P utilizando UDP com a ajuda de um servidor S conhecido, mesmo que os clientes estejam atrás de redes com NAT. O servidor de ajuda assume que os dois clientes já possuem conexões ativas com o servidor, como mostra a Figura 2.9. Sendo assim, quando um cliente se registra no servidor, o servidor guarda dois pontos de destino para esse cliente, sendo um deles o par IP/Porta (UDP) que o cliente conversa com S, representando uma rede interna e o par de IP/Porta (UDP) que o servidor tem do cliente, representando uma rede externa [30].

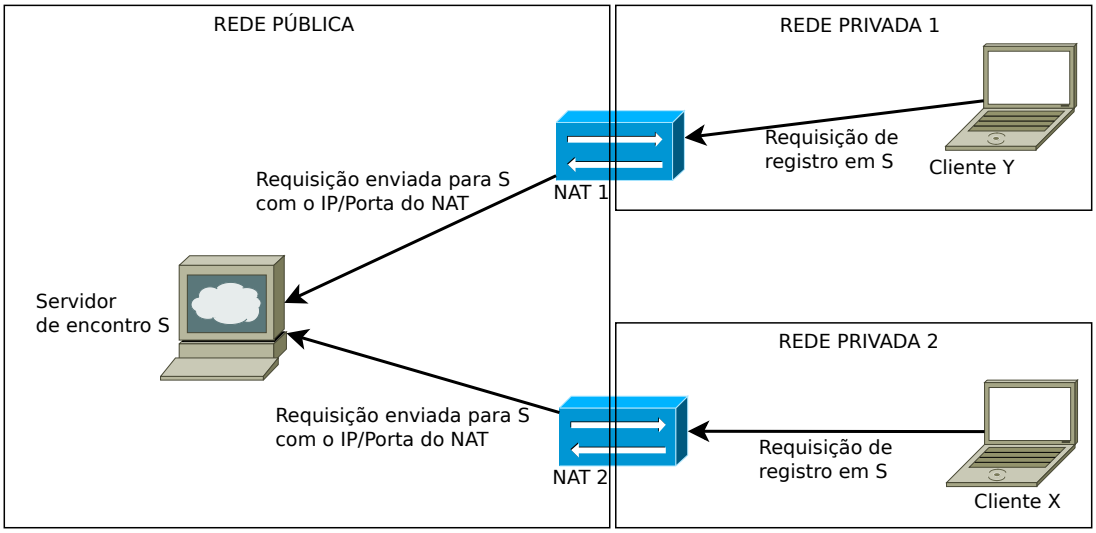

Figura 2.9: Registros dos clientes X e Y em S utilizando NATs em redes distintas.

Caso o cliente X deseja conectar-se diretamente ao cliente Y através de uma conexão UDP, utilizando o endereço externo do NAT 1. O NAT 1 irá descartar as mensagens enviadas, por não serem iguais ao IP contido no servidor S, que eram os endereços do NAT 2. Para que essa comunicação seja possível, o cliente X envia uma mensagem ao cliente Y, contendo o IP público de Y, e envia também simultâneamente uma mensagem de retransmissão a S para Y, pedindo para que Y inicie uma conexão UDP com o endereço público de X. Sendo assim, o servidor S agirá como uma introdução para o pareamento de dois clientes [30].

Já o segundo método permite que dois *hosts* atrás de NAT possam se conectar através de uma rede externa utilizando TCP. Tal descrição pode ser realizada, de forma que um cliente X queira se conectar, utilizando Pacotes de Sincronização (do inglês, *Synchronization Packet* - SYN), com um cliente Y através de uma conexão TCP, que responderá com Pacotes de Reconhecimento (do inglês, *Acknowledgement Packet* - ACK). Ambos X e Y enviarão um ao outro pacotes simultâeneos SYN e

se responderão com pacotes ACK para que a conexão seja estabelecida [31]. Para isso, ambos clientes precisam ter conexões ativas com um servidor de ajuda S, que armazena cada ponto de destino para os clientes [30].

- Predição de número de porta

  Este método é dividido em dois modos de serem realizados, um através de conexões UDP e outro através de conexões TCP. No primeiro, ocorre uma predição do próximo número de porta baseado no que foi atribuído anteriormente pelo NAT [31]. Ou seja, ele funciona analisando o comportamento do NAT, tentando adivinhar os números de portas públicas que serão atribuídas a futuras sessões. Isso somente é possível, porque a maioria dos NATs fazem atribuições de portas em sequeências [30]. Já o segundo é uma variação do TCP *Hole Punching*. Mas ele não é muito utilizado por ser mais frágil e dependente de tempos exatos da transmissão em comparação ao primeiro método. Isso acontece porque a predição da porta pública atribuída ao NAT pode estar errada. E também, se um cliente receber um pacote SYN muito rapidamante, não dando tempo do pacote do outro cliente chegar, o NAT pode rejeitar o pacote, fazendo que o NAT possa rejeitar futuras tentativas utilizando a mesma porta.

2.4.6 Vantagens e desvantagens do NAT

A implementação do NAT deu uma sobrevida ao IPv4, facilitou e agilizou a implementação de um processo convencional de *proxys*, pois agora os dispositivos dentro do NAT possuem somente um endereço externo, ao contrário de cada endereço externo para cada dispositivo [32]. Ele também possibilitou o isolamento do tráfego em uma rede, de forma que houvesse uma proteção ao dispositivo que fica "atrás" do NAT. Pois, esse isolamento, permite que um computador em uma rede externa à rede criada pelo NAT, não consiga se conectar a um computador dentro dele, por não saber seu IP. E, consequentemente, não conseguindo derrubá-lo. Por outro lado, essa mesma vantagem também trás uma desvantagem, como um dispositivo externo à rede do NAT não consegue alcançar um dispositivo interno para uma conexão ilegítima, ele também não conseguirá estebelecer uma conexão para algo que seja legítimo. Sendo assim, algumas aplicações necessitam criar subterfúrgios para elas poderem realizar essa conexão com sucesso, como o acesso a *peer-to-peer*, por exemplo. Necessitando fazer um acesso a um servidor externo (*rendezvous server*) e criar um ponteiro reverso para poder ser conectado a essa rede P2P desejada.

2.5 *Sniffers* de rede

Para fazer uma verificação dos pacotes dos protocolos e ver como está funcionando o sistema implementado, se está funcionando conforme o especificado na teoria, será utilizado neste trabalho um programa gratuito, vastamente utilizado em nível global, que analisa os pacotes das interfaces de rede do computador, denominado Wireshark. Há também outros *sniffers* de rede gratuitos, tais como o Microsoft *Network Monitor*, o Capsa *Packet Sniffer*, o NetworkMiner e o SniffPass [33].

O Wireshark analisa detalhadamente todos os pacotes que estão chegando e saindo do dispositivo que está sendo utilizado, bem como todos os pacotes que estão trafegando na rede. De acordo com o site oficial do Wireshark [34]: "... é o mais conhecido e amplamente usado analizador de protocolos de rede. Ele permite que você veja o que está acontecendo na sua rede em um nível microscópico e é de fato (e às vezes por direito) um padrão utilizado por muitas empresas comerciais e sem fins lucrativos, agências governamentais e instituições educacionais". Na Figura 2.10 é mostrada um exemplo de sua tela ao executá-lo e após "escutar" alguns pacotes.

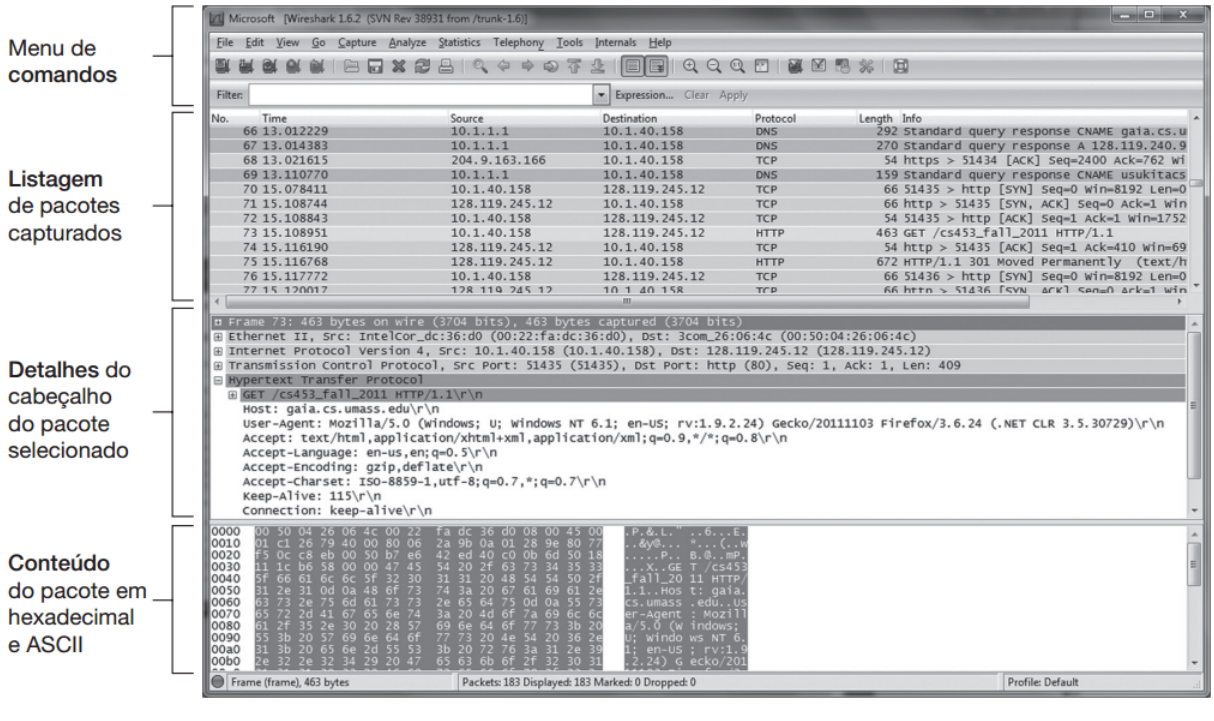

Figura 2.10: Tela de funcionamento do Wireshark [4].

Capítulo 3

Protocolos de travessia de NAT

Para solucionar o problema de conexão de um servidor para um cliente dentro de um NAT, como visto na seção 2.2.6, foi-se pensado em protocolos que pudessem realizar essa travessia. Alguns deles serão descritos detalhadamente aqui, sendo estes os mais utilizados hoje. Dentre os protocolos que serão elencados neste capítulo, o escolhido para o desenvolvimento deste trabalho será o protocolo PCP, que será utilizado para fazer a travessia de roteadores com NAT.

3.1 NAT-PMP

Este protocolo foi utilizado inicialmente em 2005, por diversos produtos da Apple, incluindo o Mac OS X, pelo Bonjour do Windows e por estações de base *wireless* de aeroportos [35]. Com o decorrer da utilização em dispositivos e da popularidade dele, foi-se então reconhecido como um padrão de RFC IETF, possuindo o número 6886 [36]. Em 2013 ele foi substituído - por ser somente indicado e utilizado em pequenas redes e para a redução significativa de mensagens de *keepalive* - pelo protocolo PCP, que realiza os mesmos procedimentos e características, mas com uma quantidade significativa de melhorias [36].

Os textos contidos nessa seção são baseados na RFC 6886 [36].

3.1.1 Funcionamento

Seu funcionamento se dá com o mapeamento de IP e porta de dispositivos dentro do NAT para que uma requisição feita através de uma rede externa possa chegar com sucesso ao destino desejado. Para isso, imagine uma rede privada atrás do NAT, em que tenham clientes com endereços IPs particulares e que estão "enxergando" o endereço IP do dispositivo NAT como seu *gateway*. Quando um pacote é enviado por um desses dispositivos

para um endereço público, ele passará pelo NAT, que olhará o IP e a porta do pacote, podendo manter a trejatória do pacote até o destino. Então o NAT cria um mapeamento, caso não exista, do endereço e porta do dispositivo que enviou o pacote para um endereço e porta externo. Sendo assim, o NAT troca o IP e a porta internos (de origem) no pacote para IP e porta externos, enviando para o próximo *gateway*. Quando um pacote é recebido da Internet, do lado "externo" do NAT, o roteador NAT irá procurar, em sua lista de mapeamentos, o IP e a porta que estão sendo utilizadas naquela transmissão. Caso uma entrada seja encontrada, ela conterá o IP e a porta interna para quem o pacote deverá ser enviado, caso contrário o pacote será descartado. Deste modo, o NAT irá trocar o endereço e porta do pacote para o endereço e porta do destinatário, transmitindo-o para esse endereço [36].

Os mapeamentos são comumente criados automaticamente como um resultado da observação dos pacotes que vão saindo, podendo também ser criados manualmente por demanda através de alguma ferramenta específica para tal, como acontece por exemplo nos roteadores de residências. Sem esses mapeamentos manuais, os clientes atrás do NAT nunca poderiam receber um pacote que seria enviado de um IP/Porta externo que tivesse ele como destino.

3.1.2 Características essenciais do protocolo

Segundo a RFC 6886 [36], o protocolo deve seguir as seguintes características:

- Cada pacote do Protocolo de Mapeamento de Portas NAT (do inglês, NAT *Port Mapping Protocol* - NAT-PMP) começa 8 bits indicando a versão, seguido por mais 8 bits indicando o opcode (código de operação)
- Todas as quantidades numéricas maiores que um *byte* são transmitidas seguindo a ordem de *bytes* da IETF, sendo o mais significativo primeiro.
- Quantidades não numéricas maiores que um *byte* são transmitidas na ordem natural dos *bytes*, sem trocas.
- Opcodes entre 0 e 127 são interpretados como requisições de clientes. De 128 a 255 são interpretados como sendo respostas de servidor, sendo que as respostas sempre contém 16 bits de tamanho, seguindo a ordem natural dos *bytes*, em que, sendo igual a zero, indicará sucesso. As respostas também contém um campo de 32 bits de inteiros sem sinal, indicando o tempo em quem o dispositivo NAT foi reiniciado ou indicando quando foi resetado o mapeamento das portas.

- O protocolo somente deve ser usado quando os clientes possuírem seu endereço IPv4 primário dentro da distância de IPv4 definidos na RFC 1918 [24] (10/8, 172.16/12, 192.168/16).
- Os clientes sempre enviam requisições para os *gateways* aprendidos através do DHCP.

3.1.3 Trocas das mensagens

Como os dispositivos que implementam NAT são geralmente de preço baixo, com memória e CPU limitados, os clientes NAT não podem ficar transmitindo várias mensagens para os *gateways*, para não haver sobrecarga, sendo assim, eles devem mandar uma de cada vez, e, uma vez que o servidor responda uma mensagem ele pode enviar uma outra novamente. Para determinar o IP externo ou uma requisição de mapeamento, o cliente envia uma requisição para a porta 5351 do roteador e espera por 250ms por uma resposta. Caso não tenha recebido uma resposta, o cliente envia uma segunda mensagem, dobrando o tempo anterior, e aguarda, por 500ms. Podendo repetir esse processo por 9 vezes, até que ou o NAT responda ou o cliente desista de realizar o mapeamento, por chegar a conclusão que o dispositivo não suporta NAT-PMP.

A partir do momento que o roteador recebe a requisição do cliente, ele fará uma resposta contendo os campos necessários, enviando-a em seguida. Uma vez recebida a resposta do servidor, o cliente precisa verificar se o IP do pacote recebido, descartando-o caso seja diferente do endereço IP de seu roteador, cuja mensagem de resquisição foi enviada.

Caso o endereço do roteador NAT mude, seja devido a reinicio, a aquisição de um novo endereço IPv4 ou qualquer outro evento que possa indicar o reinício da tabela de mapeamento, o *gateway* precisa enviar uma reposta para o endereço local de *multicast* 224.0.0.1, na porta 5350.

Para criar um mapeamento, o cliente necessita enviar um pacote UDP para a porta 5351 do roteador. Após o cliente enviar o pacote para o NAT, ele esperará uma resposta dele. Se após 250ms o cliente não tiver recebido uma resposta do roteador, ele reenvia uma requisição, conforme visto no começo da seção. Depois do roteador receber a requisição, ele enviará uma mensagem para o cliente.

Se um cliente falhar em renovar o mapeamento e se o tempo de vida do mapeamento dele expirar, então o mapeamento para ele deverá ser automaticamente deletado, podendo deletar também para um determinado cliente. Se um cliente não falhou em renovar e deseja deletar o mapeamento para ele, então ele deverá enviar uma mensagem para o

roteador requisitando-o. Assim, o roteador enviará um pacote para o cliente, confirmando a deleção.

3.2 PCP

O protocolo PCP é uma evolução do NAT-PMP. Ambos tem a mesma função, mas o PCP apresenta melhorias significativas, por incluir o suporte a IPv6, uma melhor gestão de mapeamento de saída, um melhor gerenciamento de regras de *firewall*, por possuir total compatibilidade com NAT em grande escala (com um *pool* de endereços externos) e um mecanismo de extensão para permitir futuras melhorias [36]. Ele é desenhado para ser utilizado em *Carrier-Grade* NAT, NATs residenciais, por exemplo e roteadores *Customer Premises Equipment* (que utilizam IPv6) [5].

Os textos contidos nessa seção são baseados no texto contido na RFC 6887 [5].

3.2.1 Funcionamento

O PCP fornece um mecanismo para controlar como os pacotes de entrada são encaminhados pelos dispositivos *upstream*, como os dispositivos NAT. Ele permite que os aplicativos criem mapeamentos de um endereço de IP, porta e protocolo externo para um endereço de IP, porta e protocolo interno. Ou seja, o PCP permite que um *host*, IPv4 ou IPv6, controle como os pacotes são transmitidos e traduzidos para um NAT, além de permitir que um *host* otimize as mensagens de *keepalive* de um NAT. Em que, esses mapeamentos, são necessários para comunicações de entrada bem sucedidas destinadas a máquinas localizadas atrás de um NAT ou de um *firewall* [5].

O protocolo possibilita que um *host* atrás de um NAT, por exemplo, seja encontrado por qualquer um na Internet, sem que esse dispositivo tenha que iniciar a conexão. Para que isso funcione em um NAT ou em um *firewall*, o *host* precisa, primeiro, criar um mapeamento de um endereço de IP, porta e protocolo público para si mesmo. Depois, ele tem que publicar esse endereço obtido em algum tipo de servidor de encontro ou enviar diretamente aos seus clientes o seu endereço externo (as duas formas são válidas). Assim, para que qualquer dispositivo tente acessar esse *host*, basta apenas olhar o endereço publicado no servidor de encontro ou utilizar o endereço enviado diretamente pelo *host*. A Figura 3.1 esquematiza esse funcionamento.

Em suma, uma das principais funcionalidades do PCP é a diminuição de mensagens *keepalive*. Para que isso aconteça, um *middlebox* precisa sair da sessão atual, causando falhas no aplicativo. Deste modo, será feita uma conexão com o servidor. Depois, ao longo da conexão, será checado o tempo para o envio de uma resposta PCP, de forma que seja enviado uma primeira requisição. Será verificado também caso haja perda de pacotes, em

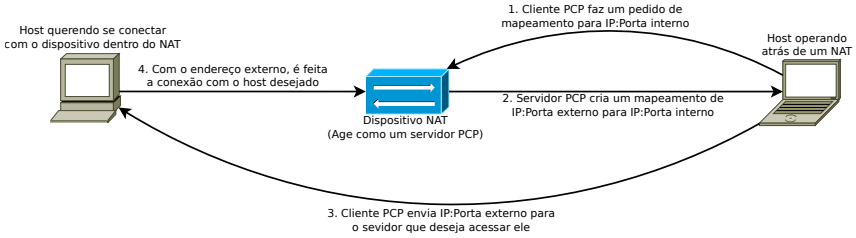

Figura 3.1: Funcionamento do PCP.

que será feita uma retransmissão dos pedidos. Também será checado se está iminente o período de encerramento da conexão, enviando uma resposta ao servidor, como também será enviado um *request* caso haja uma perda de conexão com o servidor.

3.2.2 Campos e trocas das mensagens

Todas as mensagens do protocolo são enviadas sobre UDP (indicando que o PCP não precisa ser executado através de um protocolo de transporte confiável), com o tamanho máximo de 1100 octetos (*bytes*). Todas os campos numéricos maiores que um *byte* serão definidos conforme a convenção de ordem de rede IETF, tendo o octeto mais significativo primeiro. Campos não numéricos são representados como são, sem ocorrer troca de *bytes*.

O campo de opções das mensagens de requisição e resposta, é utilizado para dizer, por exemplo, se um cliente nunca faz uma requisição de um mapeamento para um outro dispositivo externo não precisa gerar opção de THIRD_PARTY, como também um servidor que nunca implementa medidas de segurança para criar mapeamentos, também não precisa criar essa opção.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Option Code   | Reserved      |          Option Length        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                       (optional) Data                         :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figura 3.2: Mensagem indicando os campos de opções [5].

Na Figura 3.2, os campos são definidos como se seguem:

- *Option code*: correspondendo ao tamanho de 8 bits, sendo que o mais significante indica a se opção é mandatória ou opcional, contendo o valor 1 e 0, respectivamente.
- *Reserved*: contendo o tamanho de 8 bits e preenchidos com zeros.

- O campo de tamanho de opção, que indica o tamanho dos dados anexados em octetos.
- O campo de dados, que contém dados opcionais.

Para começar uma conversação e pedir uma operação de MAP ou de PEER para um servidor PCP, o cliente primeiramente precisa identificar qual servidor ele quer utilizar para realizar as transmissões de mensagens. Para isso, ele utiliza de uma lista de servidores disponíveis em um arquivo de configuração, através do DHCP ou então ele utiliza de uma lista padrão do roteador que contém os servidores PCP, podendo ser IPv4 ou IPv6, dependendo da necessidade do cliente, caso ele contenha um dos dois. Caso ele contenha ambos, ele usará um servidor para cada tipo de IP dele. Com essa informação decidida, o cliente formula a sua mensagem contendo os campos indicados na Figura 3.3, em que a porta do cliente deve ser gerada automaticamente, por conta da quantidade de possibilidades de clientes que existem na rede, evitando ambiguidade nas portas utilizadas. Ele deve incluir na mensagem o seu endereço de IP na mensagem, evitando, deste modo, que seja desperdiçado recursos com um NAT inesperado, ou *firewall*, no caminho entre o cliente e o servidor PCP. Caso seja achado um outro servidor NAT controlado por PCP (PCP-*controlled* NAT) no meio do caminho, ele realizará um mapeamento desse IP interno para o IP do servidor desejado, criando-o primeiro antes de enviá-lo, alterando o IP interno do pacote para o IP externo do PCP-*controlled* NAT, enviando-o em seguida.

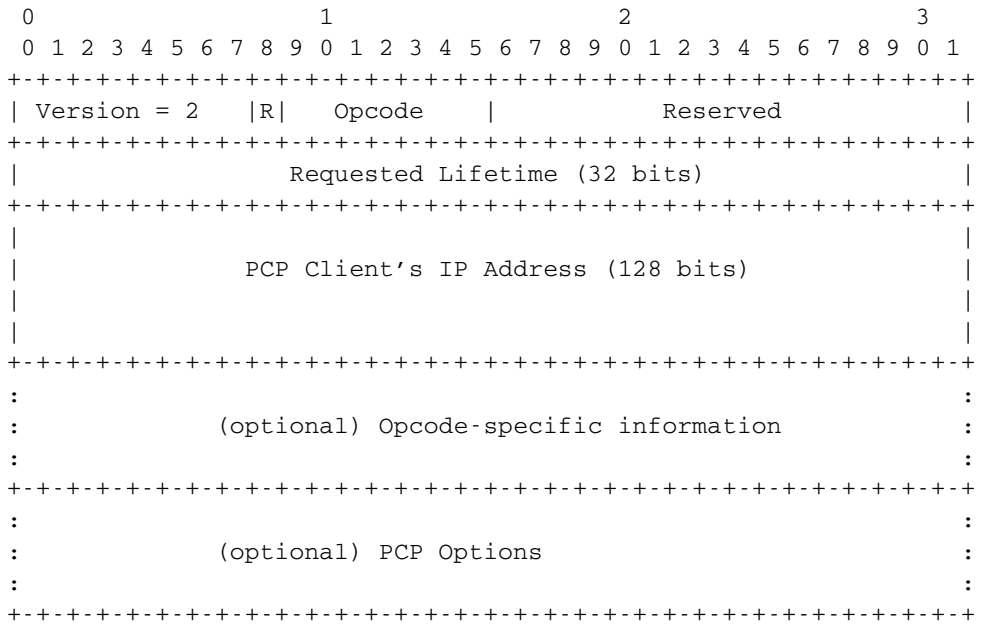

Figura 3.3: Mensagem de requisição do cliente [5].

Na Figura 3.3, os campos são definidos como se seguem:

- O campo de versão corresponde à versão 2, que foi utilizada para fazer a RFC 6887 [5].
- O campo R indica se a mensagem é de resposta ou de requisição.
- O campo opcode, que indica a operação a ser realizada.
- O campo *reserved* que deve ser zero e ignorada ao recebê-la.
- O campo de *lifetime*, que contêm um inteiro sem sinal de 32 bits, utilizado para saber o tempo de vida na hora da requisição das operações de MAP e PEER.
- O campo, de 128 bits, contendo o endereço IP do cliente, sendo no caso de um IPv4 preenchido por ::ffff:0:0/96, em que o os primeiros 80 bits são deixados como zero e os próximos 16 são definidos como 1, sobrando os últimos 32 bits para o IP.
- O campo de informações específicas de opcode.
- O campo de opções PCP.

Após receber uma requisição do cliente PCP pela mesma interface de rede, o servidor analisa a mensagem, validando-a, verificando o tamanho da mensagem recebida e seus campos, sendo maior ou menor do que o esperado, averiguando a igualdade da versão do cliente e servidor no primeiro byte da mensagem e avaliando se o endereço IP da mensagem é o mesmo endereço IP do cliente que enviou a mesma. Caso um erro seja encontrado, o servidor gera e envia uma resposta de erro para o cliente, alterando os campos de R, de *result code*, *lifetime*, *Epoch* e campos reservados e, por fim, o campo de Outros. Após validá-la, o servidor envia uma resposta ao cliente, conforme a Figura 3.4.

Na Figura 3.4, os campos são definidos como se seguem, sendo descritos somente os campos diferentes à mensagem de requisição do cliente:

- Campo de código de resultado, sendo 0 em caso de sucesso (conforme especificado todos os valores na página 19 da RFC 6887 [5]).
- O campo de *lifetime* que indica o tempo do mapeamento definido pelo servidor.
- O campo de *Epoch*, que diz o tempo do servidor (valor que é incrementado em 1 a cada segundo que se passa).
- O campo de reservado, que conterá zeros, caso a mensagem tenha sido corretamente analisada.

Após o cliente receber a resposta do servidor PCP, ele verifica se o endereço e porta da mensagem correspondem ao do servidor, destruindo silenciosamente caso não seja. Ele também fará a validação da mensagem, assim como o servidor PCP realizou, ao receber a

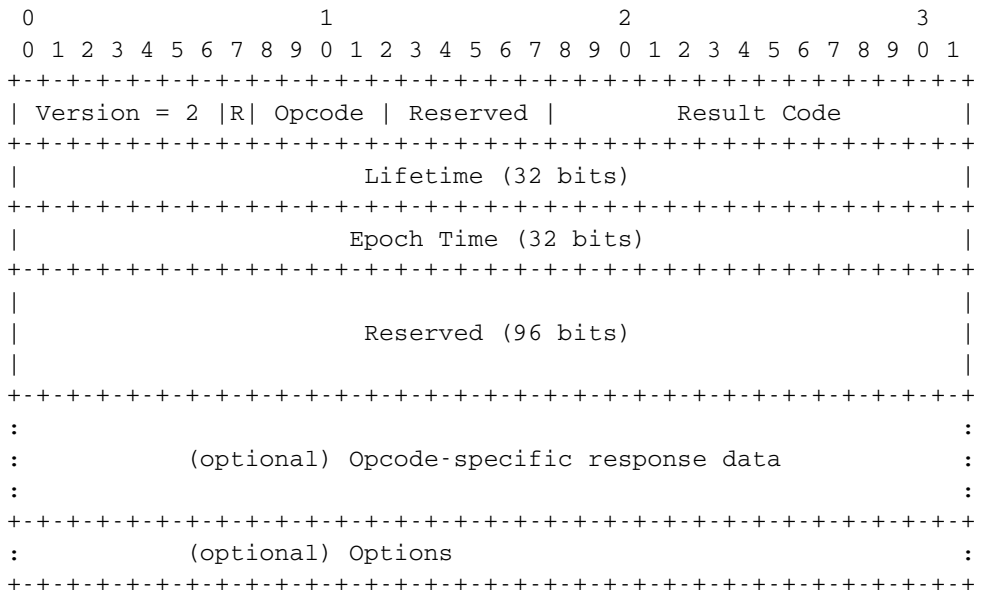

Figura 3.4: Mensagem de resposta do servidor PCP [5].

primeira mensagem do cliente PCP. Sendo assim, ele verificará os campos de R, caso não seja o valor 1, se a mensagem tem menos de 4 octetos, se a mensagem contém um código de erro, se a resposta for menor que 24 octetos, maior que 1100 octetos ou não seja múltiplo de 4 e verificará o campo de opcode se está igual ao anterior. Após a validação bem sucedida, o cliente irá chegar o campo *Epoch* para determinar se é necessário recuperar o estado do sevidor PCP e verificará também caso o campo de resultado seja 0, mostrando a requisição bem sucedida.

As mensagens de requisição de um cliente PCP podem ser de dois tipos, ou operação de MAP ou operação de PEER, sendo utilizadas em:

1. Um *host* operando um servidor e esperando por conexões de entrada
2. Um *host* operando um cliente e um servidor na mesma porta
3. Um *host* operando um cliente e esperando a otimização de mensagens de tráfego de *keepalive*
4. Um *host* operando um cliente e esperando a restauração de estado perdido do NAT.

Neste trabalho serão descritas as duas primeiras operações. Caso necessário um maior aprofundamento da funcionalidade das operações, a descrição está nas páginas 33 a 39 da RFC 6887 [5].

Quando operando um servidor, o cliente PCP sabe se ele necessita de um servidor IPv4, IPv6 ou ambos, enviando assim uma ou mais requisições de MAP a depender de sua necessidade. Caso ele tenha somente o campo de IPv4, ele poderá pedir também

um servidor de IPv4 e um de IPv6, alterando somente os campos necessários conforme descrito no início da seção. Pode também enviar uma operação de MAP somente para um servidor IPv4 ou um servidor IPv6, dependerá do uso requerido. Tendo isso, um *host* operando um servidor escuta por tráfegos em uma determinada porta, ele nunca inicia o trafégo por aquela porta. Para isso, o *host* precisa criar um mapeamento de protocolo e IP/Porta externo para si mesmo, utilizando a operação de MAP e precisa publicar esse protocolo e IP/Porta em um servidor de apoio [5].

Quando um *host* opera um cliente e um servidor na mesma porta, ele primeiro estabelece um receptor local, em seguida envia o protocolo utilizado, o IP/Porta público e privado para um servidor de apoio e inicia uma conexão externa deste endereço e porta. Em contrapartida, uma aplicação que utiliza a mesma porta para conexões externas e internas precisa sinalizar o servidor com uma operação de MAP e esperar receber uma resposta dele de que está tudo confirmado e correto, antes de realizar qualquer envio de pacotes para aquela porta.

As mensagens de requisição realizadas pelo cliente PCP possuem dois opcodes, sendo eles O MAP e o PEER. De acordo com a RFC 6887 [5], os mapeamentos criados por requisições de PCP MAP são, por definição, Mapeamentos Independentes de Ponto Final (do inglês, *Endpoint-Independent Mapping* - EIM) com Filtragens Independentes de Ponto Final (do inglês, *Endpoint-Independent Filtering* - EIF), mesmo em um NAT que normalmente cria Mapeamentos Dependentes de Ponto Final (do inglês, *Endpoint-Dependent Mapping* - EDM) ou Filtragens Dependentes de Ponto Final (do inglês, *Endpoint-Dependent Filtering* - EDF) para conexões externas, desde que o propósito de um mapeamento é de receber trafégo de qualquer *endpoint* remoto e não de um específico.

Para que uma mensagem de requisição de mapeamento seja gerada, o cliente pode definir o IP e a porta externa que ele quer, podendo ser atribuída a ele ou não, como por exemplo no caso de NATs de Grande Escala (do inglês, *Carrier-grade NAT* - CGN), em que vários dispositivos tentam mapear as portas 80, 443 e 8080, que são portas restritas e abertas a somente um dispositivo. Então, o servidor irá verificar se a porta está disponível, e, caso não esteja, irá utilizar uma outra porta para o cliente. No caso do roteador NAT encerrar inesperadamente, os campos de IP e porta externos sugeridos que foram previamente alocados para o cliente pode facilitar a alocação do servidor, pois pode ser que o cliente possa conseguir esse IP e porta como foi estabelecido antes do *crash*.

Sendo assim, as mensagens de mapeamento são transmitidas com os campos de *mapping nonce*, protocolo, reservado, porta interna, tempo de vida da requisição, porta externa sugerida, IP externo sugerido, como mostrado na Figura 3.5. Em que, caso o endereço IP/Porta externos da resposta sempre são iguais ao endereço IP/Porta internos da requisição, então a operação de MAP será intrepretada como um *firewall*, para o NAT.

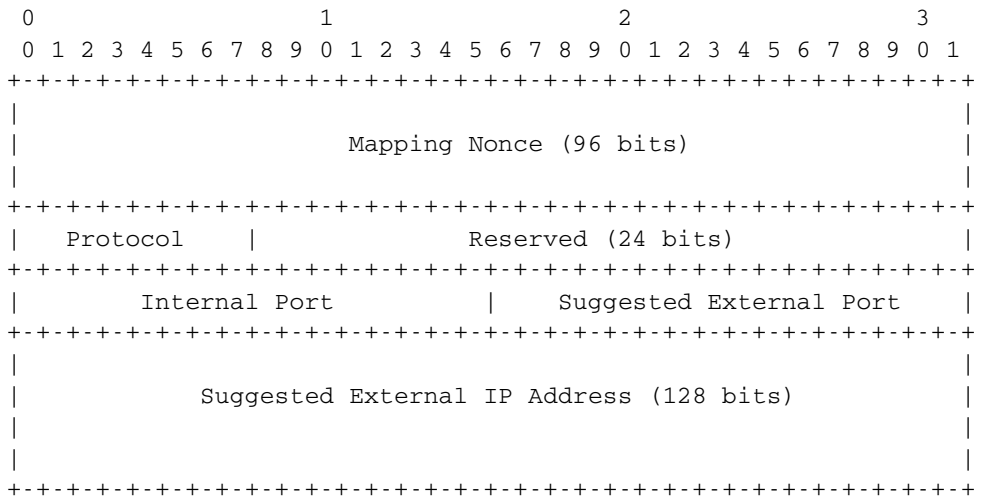

Figura 3.5: Campos da requição de mapeamento do cliente [5].

Na Figura 3.5, os campos são definidos como se seguem:

- O campo *Mapping Nonce* contém valores aleatórios gerados pelo cliente PCP e é utilizado como parte da validação das respostas pelo cliente.
- O campo de tempo de vida da requisição contém o tempo de vida do mapeamento, onde caso seja 0, será a operação de deleção do mapeamento. Esse campo fica contido no cabeçalho da mensagem.
- O campo protocolo contém o tipo de protocolo que está sendo utilizado, sendo 6 para TCP, 17 para UDP e 0 para todos os protocolos, conforme descrito nos registros da lista de protocolos da Autoridade para Atribuição de Números de Internet (do inglês, *Internet Assigned Numbers Authority* - IANA), disponível em https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml.
- O campo reservado que conterá o valor 0.
- O campo de porta interna, que contém o número da porta interna para o mapeamento.
- O campo de porta externa sugerida, que é o número que o cliente deseja que a porta externa seja alocada, utilizando 0 caso ele queira uma porta qualquer ou caso não saiba a porta exata para a sua aplicação.
- O campo de endereço IP externo sugerido, que contém o endereço IPv4 ou IPv6 que o cliente sugeriu. Este campo é útil caso o servidor PCP tenha perdido o seu estado, devendo ser 0 caso o cliente não saiba o IP externo ou não tem uma preferência pelo mesmo.

Caso um cliente deseja renovar um mapeamento para que ele continue existindo, o cliente precisa enviar uma nova operação de MAP ao servidor, indicando a porta interna requerida, deve colocar os valores de IP e porta externa que possui, nos campos de endreço IP externo sugerido e porta externa sugerida. É papel do cliente renovar os mapeamentos antes do tempo expirar, caso contrário o mapeamento será deletado do servidor.

Com o servidor recebendo a requisição da criação de um MAP pelo cliente, ele deve verificar se o campo de tempo de vida é diferente de zero, e que, caso sendo, ele verificará primeiramente os campos de protocolo e porta interna se são diferentes de zero. Se for, será realizado então um novo mapeamento ou uma renovação de tempo do mapeamento do cliente. Caso o protocolo seja diferente de zero e a porta interna seja 0, ele indicará que o cliente está querendo que o servidor realize um novo mapeamento ou estenda o seu tempo de vida para todo o tráfego para que será redirecionado para aquele protocolo podendo ser utilizadas quaisquer portas. Se a mensagem recebida conter o campo de protocolo e o campo correspondente à porta interna forem zero, o servidor criará então um mapeamento que permite que todas as requisições para todos os protocolos e todas as portas sejam redirecionados ao cliente, também conhecido como Zona Desmilitarizada (do inglês, *Demilitarized Zone* - DMZ) *host*. E, se os valores dos campos de protocolo for zero e o de porta interna for diferente de zero, então o servidor deve retornar uma mensagem de erro dizenro que a requisição está mal formada. Com o tempo de vida da mensagem igual a zero, o servidor irá deletar o mapeamento para aquele cliente. Casos de erros não abordados nesse trabalho podem ser encontrados na RFC 6887 das páginas 44 a 48.

Após fazer todas as validações pertinentes e correspondentes, o servidor PCP avaliará se o mapeamento já existe ou não, e, caso já exista ele retornará uma resposta contendo o endereço e porta externos como também o novo tempo de vida do mapeamento, renovando-o. Se o mapeamento não existir, o servidor tentará criar utilizando o endereço e porta externos sugeridos pelo cliente, observando os casos em que esse mapeamento não possa ser efetuado, conforme consta na página 47 da RFC em questão. Enviando uma resposta ao cliente. Os campos de mensagem que o servidor utilizará como resposta estão mostrados na Figura 3.6

Na Figura 3.6, os campos são definidos de forma semelhante à mensagem de requisição, alterando-se:

- O campo de porta externa atribuída, que contém o valor da porta externa dada ao cliente.
- O campo de IP externo atribuído para o mapeamento, contendo um endereço IPv4 ou IPv6.

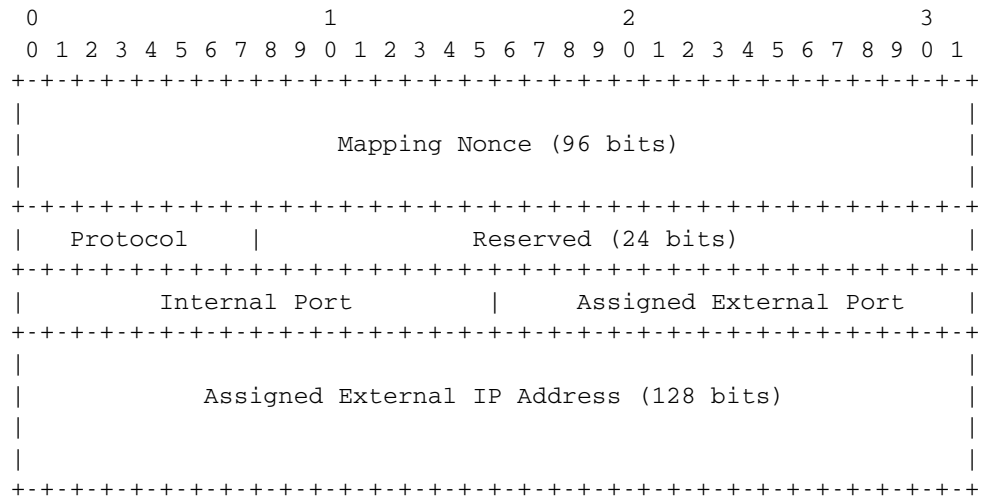

Figura 3.6: Campos da resposta de mapeamento do servidor [5].

Após o recebimento da resposta do servidor, o cliente deve comparar o endereço IP de destino, o protocolo, a porta interna e o *mapping nonce* da mensagem de requisição com a mensagem de resposta, para ver se são iguais. Sendo assim, definidos um novo mapeamento ou uma renovação de mapeamento, o cliente PCP pode utilizar o endereço IP e porta externos como necessitar, devendo criar um mecanismo que conte o tempo para a renovação do seu mapeamento. Caso a resposta do servidor seja que ele não tinha recursos no momento, o cliente PCP não deve enviar mais mensagens ao servidor para um novo mapeamento pelo limite de tempo estipulado no campo de tempo de vida.

Já a operação de PEER, que o protocolo também realiza, é responsável por fazer uma requisição de um mapeamento dinâmico de saída para um determinado *peer* da rede, ou seja, ele pede ao servidor que crie uma porta de saída para um cliente específico. E também ele pode fazer a renovação de um mapeamento para um determinado cliente. Como neste trabalho não será utilizada a operação de PEER, deixa-se-ão o entendimento dos detalhes de suas funcionalidades a interesse do leitor, contidos da página 50 a 57 da RFC 6887 [5].

3.3 STUN

De acordo com a RFC 5389 [37], o protocolo de Travessia de Sessões Utilitárias para NAT (do inglês, *Session Traversal Utilities for NAT* - STUN) foi formalizado anteriormente, no ano de 2003, através da RFC 3489 [38], mas ele foi descontinuado por apresentar vários problemas em seu funcionamento, tais como problemas de segurança graves, algumas vezes os endereços e portas aprendidos são usáveis nas comunicações P2P, mas às vezes não são, ele não ofecere uma alternativa de descobrir se o protocolo funcionava correta-

mente ou não, não indicando uma solução caso não funcionasse, entre outros problemas [37]. Sendo assim, o protocolo sofreu algumas leves alterações para que se tornasse "utilizável", tais como a mudança de protocolo que ele opera, sendo agora o TCP e sobre o UDP, ao contrário do anterior que era somente sobre o UDP, foi adicionado também uma maior extensibilidade em sua estrutura, um mecanismo de *cookie* mágico para realizar a demultiplexação do STUN em aplicações que "roubavam" 32 bits de um ID de 128 bits, entre outras alterações contidas nas páginas 45 a 47 da RFC 5389 [37]. Nesta seção, será discutido sobre o funcionamento do protocolo STUN contido na RFC 5389 [37] e todos os seus textos são baseados nessa mesma RFC.

3.3.1 Funcionamento

Este protocolo é baseado em uma situação de cliente-servidor suportando dois tipos de transações. A primeira delas é a transação de requisição/resposta, em que o cliente envia uma requisição ao servidor e o servidor retorna uma resposta. A segunda é uma transação de indicação, em que ou o cliente, ou o servidor, envia uma indicação que não gera nenhuma resposta.

Uma possível configuração do protocolo é o caso em que existem duas entidades, denominados agentes que implementam o STUN, tal que o primeiro deles, o cliente, está conectado em uma rede privada 1, que conecta em uma rede privada 2 através do NAT 1. Deste modo, a rede privada 2 se conecta na Internet através do NAT 2, que se conecta no servidor STUN contido na Internet pública, tal como mostra a Figura 3.7.

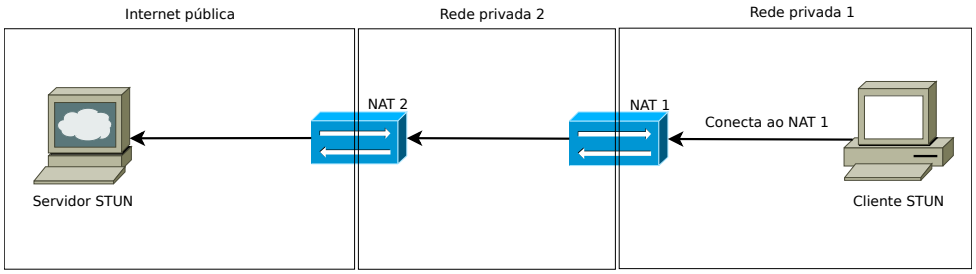

Figura 3.7: Exemplo de configuração.

Todas as mensagens do STUN começam com um cabeçalho fixo, que contém um método, uma classe e o ID da transação seguidos de zero ou mais atributos. Tal que, o método indica qual requisição, ou indicação, é, dentre vários (na RFC só trata do método *Binding*, que pode ser usado tanto na transação de requisição/resposta quanto na transação de indicação). A classe, indica se a mensagem é uma mensagem de requisição, de resposta bem sucedida, se é uma mensagem de erro ou se é uma indicação. E o ID da transação é um número aleatório de 96 bits, que serve para identificar as transações. Para

as transações de requisição e resposta, o ID é escolhido pelo cliente. Já para transações de indicação, o ID é escolhido pelo agente que envia a indicação.

3.3.2 Campos e trocas das mensagens

Essa operação de *Binding* é importante para as transações, porque na de requisição e resposta, ele é responsável por determinar a ligação do NAT alocado para o cliente. Seu funcionamento se dá através de uma requisição do cliente para o servidor STUN. Onde, quando uma requisição desse tipo chega no servidor, ele pode ter percorrido um ou vários NATs no caminho, assim como mostrado na Figura 3.7. A medida que uma mensagem de requisição *Binding* vai percorrendo os NATs, cada roteador vai trocando o endereço IP de origem e a porta para o dele nas mensagens. Sendo assim, como resultado, têm-se que o endereço IP recebido pelo servidor é o endereço IP do dispositivo NAT mais perto dele (*A.K.A. Reflexive transport address*). Por conseguinte, o servidor STUN copia esse endereço em um atributo de mapeamento utilizando a operação de XOR, armazenando em sua mensagem de resposta, enviando-a, posteriormente, de volta ao cliente. Assim, ao percorrer o caminho de volta, os roteadores contendo NATs, mudarão o IP de destino no cabeçalho da mensagem, não alterando o endereço gerado através do XOR que foi anexado à mensagem [37].

Um agente que deseja realizar uma requisição ou uma indicação ao servidor deve primeiro definir todos os campos da mensagem como visto anteriormente e indicar qual protocolo de transporte que ele usará, podendo ser UDP, TCP a Segurança da Camada de Transporte (do inglês, *Transport Layer Security* - TLS) sobre TCP e deve indicar também como que o agente determina o endereço de IP e a porta do recipiente, para depois enviá-la.

Caso o agente deseja utilizar o protocolo UDP para o envio, ele estará ciente de que a mensagem pode ser perdida em seu transporte. A confiabilidade de uma transação de requisição-resposta é realizada através de retransmissões da requisição da mensagem pela aplicação do cliente, possuindo um tempo de acordo com o Retransmition TimeOut (RTO), que dobra a cada retransmissão. Ao contrário das indicações que não são retransmitidas. Por outro lado, caso o agente deseja enviar uma mensagem através do TCP ou do TLS sobre o TCP, o cliente deve abrir uma conexão TCP para o servidor. O STUN pode ser o único protocolo sendo enviado por uma conexão, ou podem existir outros protocolos que também estejam sendo transmitidos, necessitando de uma multiplexação e sendo enviado no topo de um tipo de protocolo de frame. Se ele estiver sozinho, ele pode ser enviado sem nenhum frame adicional [37].

Se ele estiver utilizando o TLS sobre o TCP, é necessário que o TLS_RSA_WITH_AES_128_CBC_SHA (que é uma especificação de cifragem

que suporta o TLS1.0, SHA1, utilizando AES de 128 bits [37]) seja implementado pelo menos minimamente. Quando o agente recebe uma mensagem de certificação TLS, o cliente deve verificar o certificado e inspecionar o site pelo certificado e também deve verificar a identidade do servidor. Caso o certificado seja inválido o cliente não deve reenviar uma nova mensagem [37].

Na condição de encerramento do servidor, ele deve manter as conexões abertas e deixar que o cliente as feche, a não ser que a conexão tenha expirado. Transações de *Binding* são aprendidas pelos clientes, então elas permaneceram válidas e interferindo no NAT somente até que a conexão seja encerrada.

Ao receber uma mensagem de requisição, o agente primeiro checa se a mensagem obedece todas as regras dos campos dela. Finalizadas as checagens, o agente observa por algum atributo desconhecido e por atributos conhecidos, mas inesperados, ignorando-os. Depois de conferir a mensagem, o servidor formula uma mensagem de sucesso, com o mesmo método que foi enviado a ele, enviando-a novamente para o agente em forma de resposta, adicionando um endereço de IP e uma porta, ambos com XOR mapeado. Sendo esse endereço e a porta, o correspondente ao de onde veio a mensagem, no caso de uma transmissão utilizando UDP. E no caso de uma tranmissão utilizando TCP ou TLS sobre TCP, o endereço será o mesmo da conexão TCP visto pelo roteador [37].

3.4 TURN

O protocolo permite que o *host* controle as operações de retransmissão e troque pacotes com seu *peers*, utilizando essas operações. Ou seja, ele permite que um cliente se comunique com vários pares utilizando um único endereço de retransmissão. O TURN foi criado para ser usado como uma parte do ICE (uma técnica de travessia de NAT) ou podendo ser utilizado individualmente também.

Os textos contidos nessa seção são baseados no texto contido na RFC 5766 [6].

3.4.1 Funcionamento

Um *host* que esteja atrás de um NAT que deseja realizar trocas de pacotes com outros dispositivos em redes públicas, que não estão atrás de outros NATs, pode utilizar a técnica de *hole punching* na tentativa de descobrir um caminho direto de comunicação, utilizando Mapeamento Dependente de Endereço (do inglês, *Address-Dependent Mapping* - ADM) ou Mapeamento Dependente de Endereço e Porta (do inglês, *Address and Port-Dependent Mapping* - APDM). Se a troca de pacotes for realizada para dispositivos atrás de NAT, é bem provável que falharão os envios, de acordo com a RFC 5766 [6].

Quando um caminho de uma comunicação direta não pode ser achada, é necessário utilizar serviços intermediários de *hosts* para agir como um retransmissor da mensagem. Em que, essa retransmissão, geralmente fica em uma rede externa e retransmite pacotes entre dois *hosts* que estão atrás de NATs. Sendo assim, o protocolo permite que um cliente atrás de um NAT faça requisições para um servidor, que agirá como um retransmissor. O cliente pode pedir para que o servidor faça retransmissões para outros *hosts* e ele pode controlar como que elas estão sendo feitas. Ele faz isso obtendo o IP e a Porta do servidor.

Quando um dispositivo envia um pacote para o servidor, ele, por sua vez, o envia para o cliente. E, quando um cliente envia um pacote ao servidor, ele retransmite esse pacote para o par apropriado utilizando seus endereços de IP e porta como o do cliente. Mas, ao usar o servidor para permitir a comunicação entre dois *hosts* atrás de NATs, acaba que o custo para o provedor do servidor ser bem elevado, pois é necessária uma largura de banda alta para a conexão com a Internet. Deste modo, é melhor que esse método seja utilizado somente quando uma conexão direta não pode ser encontrada.

3.4.2 Trocas de mensagens

Como o TURN é uma extensão do STUN, os campos e as mensagens são identicas aos dos dois protocolos, não especificando assim esses campos na RFC 5766 [6]. Apenas uma mensagem se difere da Seção 3.3.2, que é a de canais de dados (*ChannelData*). Em uma configuração comum do protocolo, um cliente está conectado em uma rede privada atrás de um ou mais NATs e um servidor está conectado em uma rede pública, na Internet. Em quaisquer outro lugar da Internet exitem *peers* que os clientes querem realizar a conexão, no qual podem estar conectados ou não atrás de um ou mais NATs. Deste modo, o cliente utilizará do servidor como um retransmissor dos pacotes recebidos por ele para o envio a esses destinos, como também um retransmissor para pacotes que vêm dos *peers* para o cliente, como mostra a Figura 3.8, onde o cliente e o servidor estão separados por um NAT, com o cliente do lado privado e o servidor do lado público.

A partir dessa configuração, um ou mais clientes devem aprender o endereço do servidor (IP e porta), através de arquivos de configuração, por exemplo, para então enviar uma mensagem, contendo seus endereços de IP e porta do *peer* desejado para o IP/Porta do servidor. Mensagem essa, que contém comandos para criar ou manipular alocações no servidor. Uma alocação nada mais é que uma estrutura de dados no servidor que contém o endereço de transporte da transmissão para a alocação, o tempo de vida da operação, entre outros atributos, como pode ser visto na Figura 3.9. Em que, esse endereço de transporte é o endereço utilizado pelo servidor para que os *peers* possam utilizar para retransmitir a mensagem ao cliente.

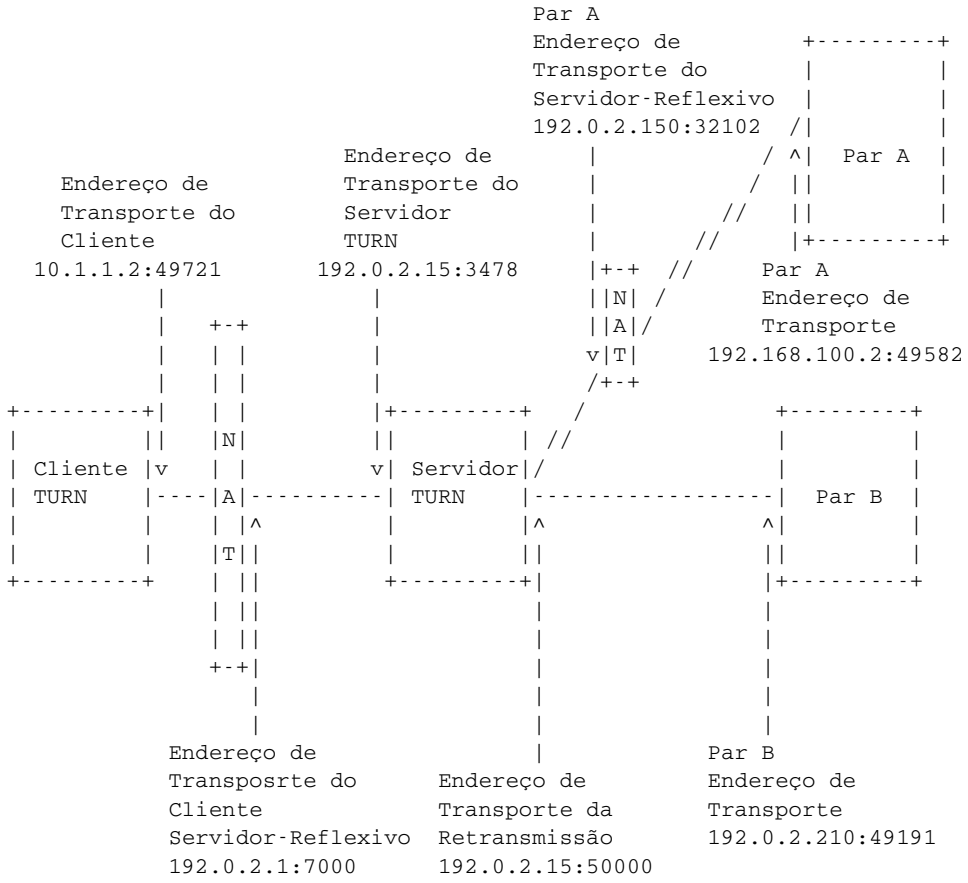

Figura 3.8: Exemplo do uso do protocolo [6].

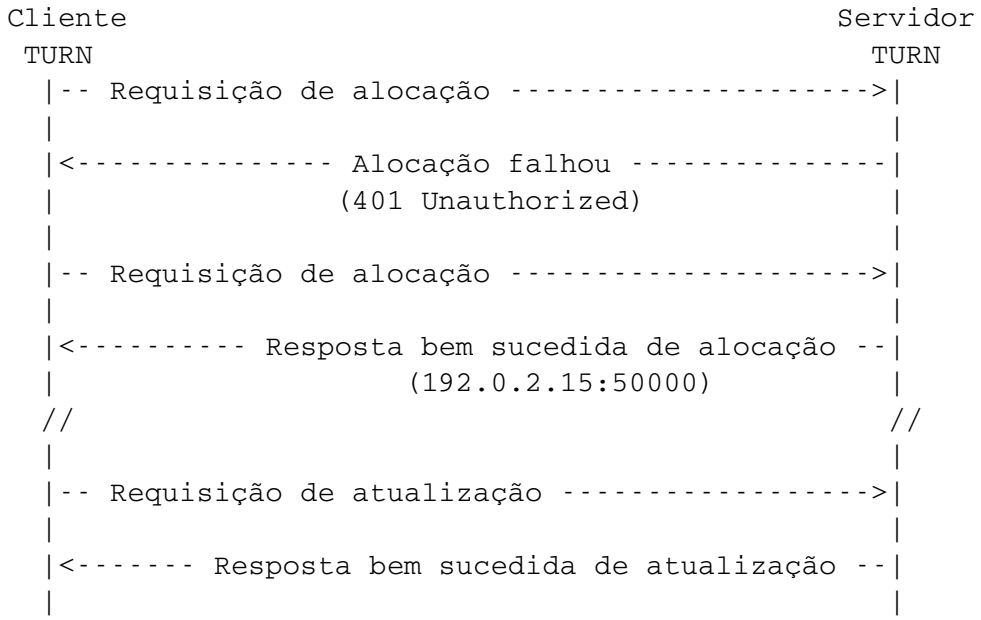

Figura 3.9: Troca de mensagens entre cliente e servidor para a alocação.

Uma vez que uma alocação é criada, o cliente pode enviar a mensagem ao servidor, observando sempre o tempo de término dela, enviando requisições de atualizações de tempo para o servidor caso necessário, com o servidor enviando, em seguida, quanto tempo a alocação ainda possui. Na mensagem que o cliente enviará, conterão os dados da aplicação que está sendo executada e uma indicação, que possui o endereço para qual *peer* que ela será enviada. Chegando no servidor, os dados serão extraídos da mensagem e, após, ele enviará uma mensagem ao cliente confirmando ou rejeitando a alocação, para depois enviar os dados ao destino solicitado através de um datagrama UDP. O servidor "enxergará" o endereço fonte desse pacote que chegou do cliente como sendo o do próprio cliente, caso não esteja atrás de um NAT, ou como o endereço externo do NAT, caso esteja atrás de um.

Do lado contrário, um par que deseja enviar uma mensagem (contendo os dados da aplicação) para o cliente, fará a transmissão da mensagem para o servidor, no endereço de transporte da alocação da mensagem que chegou anteriormente como endereço de destino, através de um datagrama UDP. Sendo assim, o servidor receberá essa mensagem e inserirá esses dados, em seguida, em uma nova mensagem, enviando-o para o cliente dizendo também qual o *peer* que enviou essa mensagem para ele. Dado que uma mensagem STUN sempre contém uma indicação de qual *peer* o cliente quer se comunicar, o cliente pode utilizar uma única alocação para se comunicar com vários pares. Cada alocação no servidor pertence a um único cliente e contém somente um endereço de transporte que será utilizado por ela, mas um cliente pode ter várias alocações no mesmo servidor ao mesmo tempo.

Caso o *peer* esteja atrás de um ou vários NATs, o cliente precisa especificar o endereço de transporte do par como o endereço do NAT que ele alcançará, para que o NAT fique encarregado de fazer a travessia corretamente para o *peer*, como por exemplo o endereço da Figura 3.10, em que caso o cliente queira enviar uma mensagem ao *Peer* A, ele deve especificar o endereço de destino como sendo o endereço de transporte do NAT que está à frente do *peer* e não o endereço efetivo do par, ou seja, deverá enviar para o 192.0.2.150:32102 ao invés de enviar para o 192.168.100.2:49582.

Os protocolos de transporte de mensagens entre cliente e servidor podem ser tanto UDP, como TCP e TLS sobre TCP. O cliente que decide qual protocolo utilizar, dependendo do nível de segurança da mensagem, e se possui *firewalls* que bloqueiam serviços UDP, por exemplo. Já para a conversação entre o servidor e um determinado *peer*, deve ser feito única e exclusivamente através do protocolo UDP, como mostra a Figura 3.10.

As mensagens de *ChannelData*, citadas no começo da seção, são mensagens contendo um cabeçalho de 4 *bytes* que contém o número do canal, ao invés dos cabeçalhos utilizados no protocolo STUN. Cada número de canal em uso é ligado a um *peer* específico, servindo

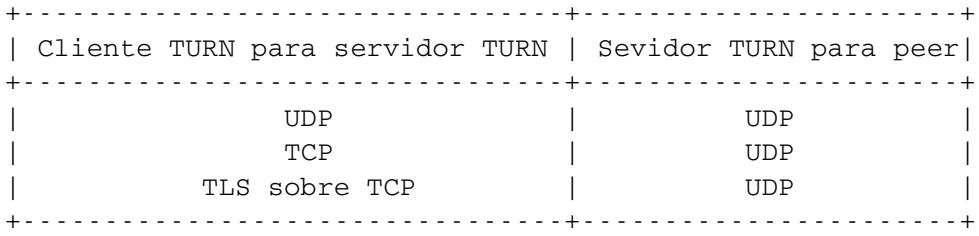

Figura 3.10: Casos de uso dos protocolos de transporte [6].

como um atalho para o endereço de transporte. Essas mensagens servem para diminuir a largura de banda necessária para uma transmissão de Voz sobre o Protocolo de Internet (do inglês, *Voice over Internet Protocol - VoIP)*, por exemplo, pois uma mensagem de indicação contém 36 *bytes* de cabeçalho, necessitando, a medida que a transmissão esteja sendo realizada, uma melhor largura de banda para o envio de várias mensagens sequencialmente. Para atribuir um canal a um *peer*, o cliente envia uma requisição de *ChannelBind* para o servidor, incluindo um número de canal ainda não utilizado. Uma vez que é realizada a ligação, o cliente pode utilizar as mensagens de *ChannelData* para enviar os dados da aplicação para o servidor, como pode ser visto na Figura 3.11, seguindo os mesmos procedimentos descritos anteriormente. Caso o cliente não utilize canais em suas mensagens, ele poderá enviar as mensagens normalmente, precisando apenas de uma autenticação válida para que as mensagens possam seguir, como visto o caso de falha na Figura 3.9, em que a mensagem do cliente não foi autenticada por não estar conforme o mecanismo de credencial *long-term*, que utiliza um login e uma senha, conforme consta na RFC 5389 [37].

3.5 Outras propostas de travessia NAT

Segundo Irvine (2010), existem outras formas que sejam capazes de realizar a travessia de NAT, um deles é o *Universal Plug and Play* (UPnP) puro, que realiza uma negociação com o roteador. Mas, essa utilidade não é segura, porque um mapeamento só irá ser encerrado quando o roteador for reiniciado ou quando o cliente pedir para que o mapeamento seja desfeito, ou seja, o cliente terá um mapeamento para sempre, não acontecendo essas condições [39]. E, também por esse motivo, será utilizado nesse trabalho a tecnologia UPnP juntamente com o PCP.

Já um outro protocolo, denominado ICE, possui um gargalo que limita muito o seu uso em aplicações que demandam uma maior agilidade e confiança dos dados, pois, de acordo com Mäenpää (2013), a cada requisição e resposta completas e bem sucedidas de mapeamento, o protocolo envia até 100 mensagens, criando um enorme atraso no

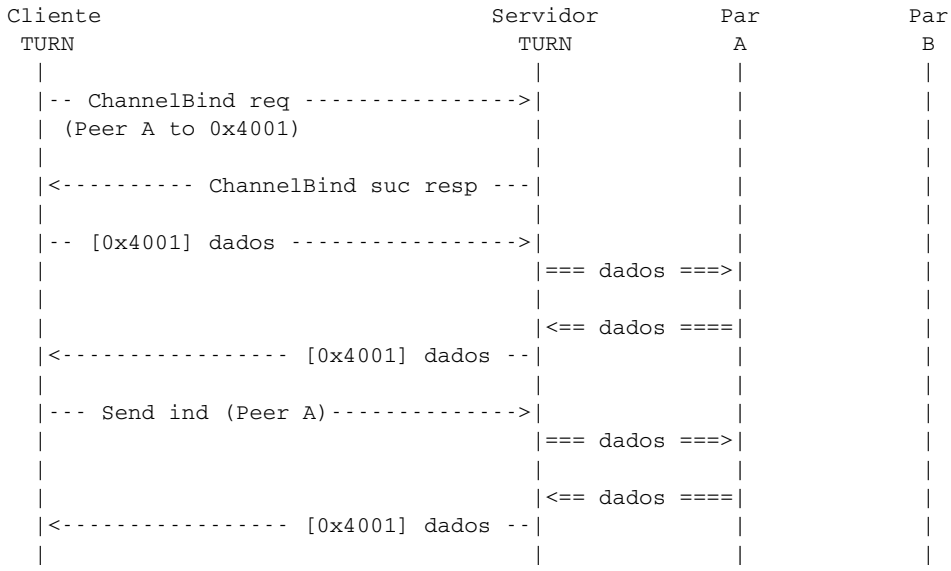

Figura 3.11: *Handshake* das mensagens de canais [6].

estabelecimento desse mapeamento requerido, chegando a um tempo de espera de 30 segundos, sendo necessários 2 tentativas para que pudesse ser criado o mapeamento com sucesso [40].

Para Kantola (2015), o Protocolo de Iniciação de Sessão (do inglês, *Session Initiation Protocol* - SIP) também não é recomendável quando utilizado juntamente com o protocolo STUN, pois o tempo de cada mensagem de *keepalive* é muito curto, levando em conta dispositivos que utilizam baterias. Como por exemplo os celulares, pois, ao encurtar esse tempo, haverá uma quantidade significativa de pacotes sendo enviados, necessitando de muito mais bateria do que com o uso do SIP com *Carriage Return - Line Feed* - CRLF, que possui um tempo de mensagem de *keepalive* fixo em 840, para dispositivos que utilizam de bateria para seu funcionamento, conforme descrito na RFC 5626 [41]. Apesar disso, esse protocolo é mais indicado para comunicações que utilizam VoIP.

Apesar de todas propostas apresentadas neste capítulo, da facilidade em encontrar-se textos referenciais sobre a maioria dos protocolos citados e seus usos em geral, neste trabalho será abordada a implementação prática e teórica do protocolo PCP, por não possuir variadas bibliografias sobre o seu uso e sobre sua implementação, a fim de incrementar a sua popularidade. Atualmente, este protocolo somente é implementado em ambientes com CGNs, que possuem um alto custo monetário, passando a faixa dos 20 mil reais. Na implementação realizada, foi-se utilizado um equivalente a 300 reais, somente o necessário para a compra do raspberry pi, de um teclado, de um mouse e um adaptador de rede sem fio.

Capítulo 4

Adaptação e Avaliação do PCP

O objetivo deste capítulo é avaliar o PCP na prática, tendo como base os conceitos e definições descritos no capítulo anterior. Para isso, foi montado um ambiente, descrito na próxima seção, para testes. Há também uma descrição das implementações dos servidor e cliente PCP com suas respectivas limitações. Após essas explicações, em 4.2, utilizando o ambiente previamente montado, foram feitos experimentos com o objetivo de mostrar o comportamento do PCP na prática.

4.1 Ambientes e implementação do PCP

Para validação do PCP como solução para a travessia de NAT, foi montado um ambiente de testes com dois computadores e um roteador, como mostrado na Figura 4.1.

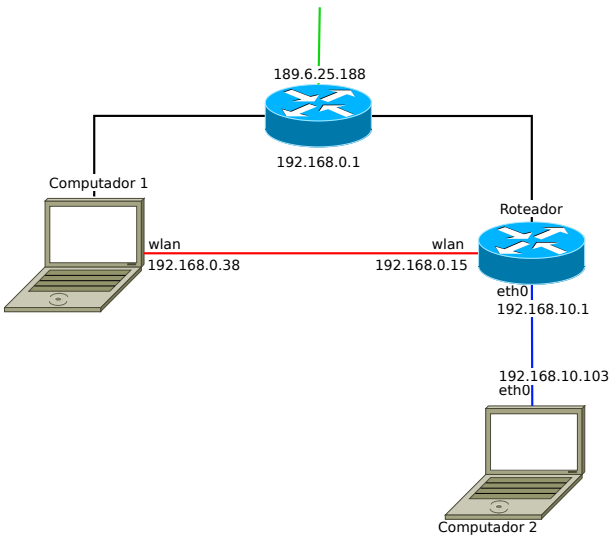

Figura 4.1: Ambiente interno (NAT) e ambiente externo (Internet).

No ambiente montado, o computador 1 e o roteador estão na mesma rede local e se comunicam pelas suas respectivas interfaces sem fio. Na figura, seu nome está como *wlan*. Já o computador 2, se encontra na subrede do roteador em questão. Eles se comunicam pela interface cabeada, chamada de *eth0*. Deste modo, é impossível que o computador 1 inicie uma comunicação com o computador 2 sem algum mecanismo de travessia de NAT implementado.

Nos testes, o roteador em questão é desempenhado por um Raspberry PI 2 Modelo B, com o sistema operacional Raspian versão 9.4 instalado e com um adaptador WiFi Ralink RT5370 conectado. Simulou-se um roteador semelhante aos modelos reais utilizados em residências, caracterizando-se, portando, em um servidor de DNS, um servidor DHCP, utilizando o protocolo NAT e por serviços de redirecionamento. Foram utilizadas as seguintes ferramentas para os protocolos:

- ISC DHCPd para a implementação de um servidor DHCP.
- bind9 para a implementação de um servidor DNS.
- iptables para a configuração do NAT e para a configuração do redirecionamento de portas.

Além disso, no roteador, foi configurado também um servidor PCP com o uso da ferramenta Miniupnp [42]. A instalação e a configuração do roteador, estão descritas no Anexo I. A decisão de uso do Raspberry PI, em contrapartida à máquina virtual VirtualBox, se deu ao fato de que o protocolo PCP não conseguia transmitir as mensagens corretamente de um cliente para o servidor e vice-versa, porque a tabela NAT da máquina virtual era inacessível para o servidor PCP. Desta forma, não era possível fazer uma abertura de portas bem sucedida através da ferramenta iptables, não possibilitando uma conexão entre os dois *hosts* da rede almejada.

No computador 2, foi instalado um cliente PCP, ao contrário do computador 1, em que não foi necessária a instalação de nenhuma aplicação relacionada à travessia de NAT. Na próxima seção, haverá uma descrição mais detalhada sobre as ferramentas PCP utilizadas no ambiente.

4.1.1 Cliente PCP

A implementação do cliente PCP neste trabalho é baseada em um projeto do Peter Tatrai [43].

Sua implementação contém as funcionalidades básicas contidas na RFC 6887 [5], como por exemplo, conseguir fazer uma requisição de mapeamento e de *peer*, contendo todos os campos que estas requisições devem suportar, como pode ser visto na Tabela 4.1, de

exemplo de funcionamento do protocolo, onde há um pedido de mapeamento da porta 9102 pelo cliente, o qual foi realizado com sucesso. Na Figura 4.2 é mostrada uma mensagem validada de uma operação de PEER, mas ela não será utilizada no roteador construído no próximo capítulo, sendo apresentada nesta seção somente para mostrá-la funcionando corretamente.

Tabela 4.1: Comparativo dos dados obtidos do pacote de requisição de mapeamento.

Campos	Valores obtidos	Significado
Versão	2	Será validado posteriormente pelo servidor
R	0	Realização da operação de requisição
Opcode	1	Indicação da operação de MAP
Requested lifetime	86400	Valor padrão do tempo de vida do mapeamento
Client IP Address	192.168.10.103	Endereço IPv4 do cliente
Mapping Nonce	140420a11df51e955c9c6d6c	Gerado aleatóriamente pelo cliente para validação
Protocol	6	Diz que o protocolo utilizado é o TCP
Reserved	0	Definido por padrão
Internal Port	9102	Porta interna requerida pelo cliente para o mapeamento
Suggested External Port	0	A escolha da porta ficará por conta do servidor
Suggested External IP Address	0	Não definição de um endereço IP externo por não conhecimento do cliente ou por não ter preferência

Ambas mensagens possuem os campos corretamente citados e valorados conforme consta a RFC em questão. Como a principal operação utilizada foi a de mapeamento, relacionou-se os campos contidos na requisição de um mapeamento, vistos na Tabela 4.1, com os campos esperados, vistos na Seção 3.3. Deste modo, é possível notar que uma mensagem de mapeamento do cliente pode ser enviada com sucesso, confirmando o funcionamento dessa operação no cliente.

O cliente implementado possui algumas limitações em relação à sua descrição contida na RFC 6887 [5]. Ele não permanece esperando por mensagens do servidor PCP, após o mapeamento ser feito com sucesso. Sendo assim, se há alguma mudança em algum

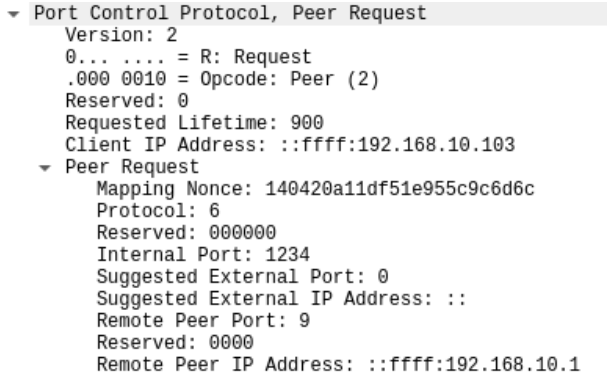

Figura 4.2: Pacote de requisição PEER.

mapeamento feito por parte do servidor, o cliente não é capaz de ser notificado e agir de acordo. Uma outra limitação encontrada é o não envio de pacotes de renovação de mapeamento de tempos em tempos.

Basicamente o cliente é capaz de identificar se há servidores PCP na rede, pedir um mapeamento preenchendo os campos que o usuário achar necessário e esperar por uma resposta do servidor, encerrando-o. Como ele não continua sua execução após pedir uma operação de MAP ou de PEER, foi necessário definir um tempo total de vida fixo para o mapeamento. Em que, no experimento, foi definido com o prazo de um dia, para que o mapeamento não expire durante a realização dos testes.

4.1.2 MiniUPnP

O MiniUPnP é um *software* livre que oferece suporte para protocolos de travessia de NAT como o NAT-PMP, PCP, STUN e TURN [42]. Existem tanto o cliente, sendo o miniupnpc, e o servidor, sendo o miniupnpd. Ambas utilizações implementam a arquitetura e a tecnologia Universal Plug and Play for Internet Gateway Device (UPnP IGD), desenvolvida pela *Open Connectivity Foundation*, que, de acordo com o próprio site da empresa, ele permite que os dispositivos se conectem e simplifiquem a implementação da rede nos ambientes domésticos e corporativos. O UPnP oferece conectividade de rede P2P generalizada, além de ser uma arquitetura aberta e distribuída, que aproveita o TCP/IP para permitir uma rede ininterrupta [44].

Neste trabalho, será utilizado o miniupnpd para responder às requisições de mapeamento enviadas pelo cliente PCP. O miniupnpc não foi escolhido para a sua utilização por ele não possuir, até a data de realização deste trabalho, a implementação do PCP.

Na Figura 4.3, para exemplo de funcionamento do protocolo, é possível ver uma mensagem de requisição de mapeamento, feita pelo cliente, através da porta 9102. Seus campos

estão detalhados na Seção 3.2.2 deste trabalho.

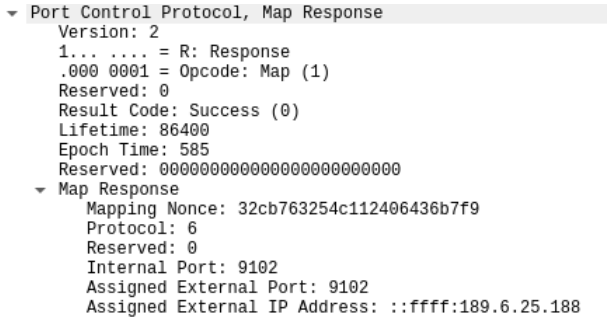

Figura 4.3: Pacote de resposta à requisição de mapeamento da porta 9102.

Esse pacote possui, primeiramente, o campo de versão, contendo o valor 2, representando a versão que o PCP está rodando. Esse valor deve ser igual ao valor dado na mensagem de requisição, gerando uma mensagem de erro, caso não seja. No campo seguinte, possui o valor de R, sendo 1, mostrando que a mensagem é do formato de uma reposta. Após, apresenta-se o campo de Opcode com o valor de 1, que indica que a mensagem é do tipo MAP e não de PEER. Seguindo no modelo visto na seção em que é descrita, o valor do campo *Reserved* é de 0, conforme consta a especificação do protocolo, sendo assim ignorado esse campo. Posteriormente, é definido o campo de *Result Code*, contendo o valor 0, indicando que o mapeamento foi realizado com sucesso, sendo 1 caso tenha gerado algum erro. Logo após, contém o campo de *Requested Lifetime*, conforme a RFC 6887 [5], em que o tempo de vida do mapeamento é definido como 86400 segundos. Em seguida, pode-se observar o tempo de *Epoch*, que indica o tempo do servidor, em quanto tempo o servidor está ativo, possuindo o valor de 585 segundos. Depois, há o campo de *Reserved* novamente, que também contém o valor de 0, mostrando que a mensagem foi corretamente analisada. Nos campos de resposta do MAP, há os valores do *Mapping Nonce*, contendo valores aleatórios gerados pelo cliente ao enviar a mensagem, para fins de validação. Seguidamente, existe o campo de Protocolo, que define o 6 valor, que indica que o protocolo que está sendo utilizado é o TCP. Logo depois, está contido outro campo de *Reserved*, que precisa conter todos os valores zerados, como define o protocolo. No campo seguinte, é apresentado o valor da porta interna para o mapeamento, possuindo o valor 9102, que foi o valor requisitado pelo cliente. Após, contém o valor 9102 no campo de porta externa atribuída, que mostra que o servidor conseguiu alocar exatamente a porta que o cliente estava pedindo. E, por último, contém o valor do IP externo atribuído para o mapeamento, contendo o valor de 189.6.25.188, mostrando o sucesso da resposta.

4.2 Experimentos e resultados

Para a realização do experimento, executou-se um servidor TCP no computador 2 (192.168.10.103), com o objetivo de fazer o computador 1 (192.168.0.38) iniciar uma comunicação com este servidor, e, assim, confirmar o PCP como sendo uma possível solução para o problema de travessia de NAT.

Para isso, primeiramente, foi executado o servidor TCP no computador 2, na porta 5005. Após, foi feita uma requisição ao servidor PCP para mapear a porta 5005 do cliente, como pode ser observado na Figura 4.4. Todos os campos das mensagens da figura podem ser visualizados e comparados com a Seção 3.2.2.

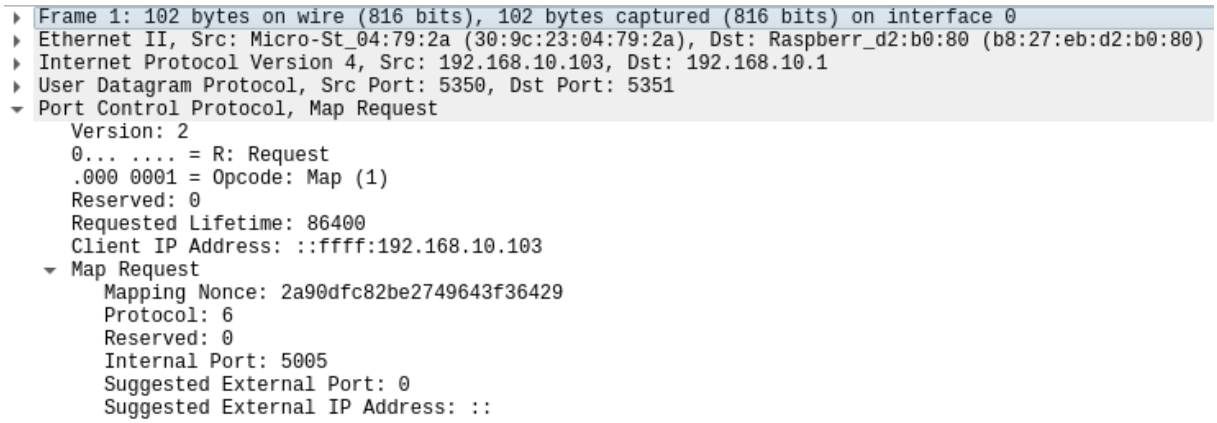

Figura 4.4: Pacote de requisição de mapeamento da porta 5005.

Na Figura 4.5, é possível ver a resposta do servidor à requisição de mapeamento feita pelo cliente. No caso, o servidor mapeou a porta 5005 do cliente para sua própria porta 5005.

Feito o mapeamento, o computador 2 pode publicar o endereço externo do seu servidor TCP, que possui o endereço de IP 192.168.0.15 e porta 5005. Sendo assim, o computador 1 pode se comunicar normalmente com o servidor no endereço publicado anteriormente, como pode ser visualizado na Figura 4.6. Nela, há a troca de mensagens entre o cliente e o servidor TCP. A captura foi feita no próprio roteador, sendo possível assim, observar como a tradução de endereços é realizada.

Na Figura 4.7, há um diagrama explicitando a troca de mensagens entre o cliente e servidor TCP. Nela, no passo 1, as mensagens são enviadas pelo cliente TCP, cujo endereço de IP é 192.168.0.38 e porta 40662, para o IP 192.168.0.15 e porta 5005, que, corresponde ao endereço do roteador. Deste modo, surge então a necessidade do roteador repassar essas mensagens ao destinatário correto, o servidor TCP, que é apontado pelo passo 2 na figura. Para isto, ele apenas substitui o IP 192.168.0.15 e porta 5005, que estão

```
▸ Frame 2: 102 bytes on wire (816 bits), 102 bytes captured (816 bits) on interface 0
▸ Ethernet II, Src: Raspberr_d2:b0:80 (b8:27:eb:d2:b0:80), Dst: Micro-St_04:79:2a (30:9c:23:04:79:2a)
▸ Internet Protocol Version 4, Src: 192.168.10.1, Dst: 192.168.10.103
▸ User Datagram Protocol, Src Port: 5351, Dst Port: 5350
▾ Port Control Protocol, Map Response
    Version: 2
    1... .... = R: Response
    .000 0001 = Opcode: Map (1)
    Reserved: 0
    Result Code: Success (0)
    Lifetime: 86400
    Epoch Time: 194
    Reserved: 000000000000000000000000
  ▾ Map Response
      Mapping Nonce: 2a90dfc82be2749643f36429
      Protocol: 6
      Reserved: 0
      Internal Port: 5005
      Assigned External Port: 5005
      Assigned External IP Address: ::ffff:189.6.25.188
```

Figura 4.5: Pacote em resposta à requisição de mapeamento.

```
 2 3.420066242  192.168.0.38    192.168.0.15     TCP  76 40662 → 5005 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3315914662 TSecr=0 WS=128
 3 3.420368220  192.168.0.38    192.168.10.103   TCP  76 40662 → 5005 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSval=3315914662 TSecr=0 WS=128
 4 3.420766188  192.168.10.103  192.168.0.38     TCP  76 5005 → 40662 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3608658227 TSecr=3315914662 WS=128
 5 3.420865407  192.168.0.15    192.168.0.38     TCP  76 5005 → 40662 [SYN, ACK] Seq=0 Ack=1 Win=65160 Len=0 MSS=1460 SACK_PERM=1 TSval=3608658227 TSecr=3315914662 WS=128
 6 3.422361289  192.168.0.38    192.168.0.15     TCP  68 40662 → 5005 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3315914664 TSecr=3608658227
 7 3.422430820  192.168.0.38    192.168.10.103   TCP  68 40662 → 5005 [ACK] Seq=1 Ack=1 Win=29312 Len=0 TSval=3315914664 TSecr=3608658227
 8 3.422788163  192.168.0.38    192.168.0.15     TCP  69 40662 → 5005 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=1 TSval=3315914664 TSecr=3608658227
 9 3.422844934  192.168.0.38    192.168.10.103   TCP  69 40662 → 5005 [PSH, ACK] Seq=1 Ack=1 Win=29312 Len=1 TSval=3315914664 TSecr=3608658227
10 3.423262537  192.168.10.103  192.168.0.38     TCP  68 5005 → 40662 [ACK] Seq=1 Ack=2 Win=65280 Len=0 TSval=3608658229 TSecr=3315914664
11 3.423404933  192.168.0.15    192.168.0.38     TCP  68 5005 → 40662 [ACK] Seq=1 Ack=2 Win=65280 Len=0 TSval=3608658229 TSecr=3315914664
12 3.423265610  192.168.10.103  192.168.0.38     TCP  69 5005 → 40662 [PSH, ACK] Seq=1 Ack=2 Win=65280 Len=1 TSval=3608658229 TSecr=3315914664
13 3.423554464  192.168.0.15    192.168.0.38     TCP  69 5005 → 40662 [PSH, ACK] Seq=1 Ack=2 Win=65280 Len=1 TSval=3608658229 TSecr=3315914664
14 3.430708408  192.168.0.38    192.168.0.15     TCP  68 40662 → 5005 [ACK] Seq=2 Ack=2 Win=29312 Len=0 TSval=3315914672 TSecr=3608658229
15 3.430783044  192.168.0.38    192.168.10.103   TCP  68 40662 → 5005 [ACK] Seq=2 Ack=2 Win=29312 Len=0 TSval=3315914672 TSecr=3608658229
16 3.431179918  192.168.0.38    192.168.0.15     TCP  68 40662 → 5005 [FIN, ACK] Seq=2 Ack=2 Win=29312 Len=0 TSval=3315914673 TSecr=3608658229
17 3.431236272  192.168.0.38    192.168.10.103   TCP  68 40662 → 5005 [FIN, ACK] Seq=2 Ack=2 Win=29312 Len=0 TSval=3315914673 TSecr=3608658229
18 3.431676740  192.168.10.103  192.168.0.38     TCP  68 5005 → 40662 [FIN, ACK] Seq=2 Ack=3 Win=65280 Len=0 TSval=3608658238 TSecr=3315914673
19 3.431738563  192.168.0.15    192.168.0.38     TCP  68 5005 → 40662 [FIN, ACK] Seq=2 Ack=3 Win=65280 Len=0 TSval=3608658238 TSecr=3315914673
20 3.434618401  192.168.0.38    192.168.0.15     TCP  68 40662 → 5005 [ACK] Seq=3 Ack=3 Win=29312 Len=0 TSval=3315914675 TSecr=3608658238
21 3.434674755  192.168.0.38    192.168.10.103   TCP  68 40662 → 5005 [ACK] Seq=3 Ack=3 Win=29312 Len=0 TSval=3315914675 TSecr=3608658238
```

Figura 4.6: Troca de mensagens entre o cliente e o servidor TCP.

configurados como o endereço do destinatário para o IP 192.168.10.103 e porta 5005 do servidor TCP real. Isto pode ser observado na Figura 4.6 onde todo pacote com endereço IP de origem 192.168.0.38 e endereço IP de destino 192.168.0.15 tem uma duplicata com mesmo endereço de origem, e com endereço IP de destino 192.168.10.103.

O mesmo acontece nos passos 3 e 4, quando a comunicação ocorre de maneira oposta, porém a substituição é feita no campo de remetente da mensagem. No passo 3 do diagrama, o pacote de confirmação é enviado pelo servidor TCP, cujo endereço de IP é 192.168.10.103 e porta 5005, para o endereço de IP 192.168.0.38 e porta 5005, que corresponde ao cliente TCP. Quando este pacote chega ao roteador, há a troca do endereço de origem pelo endereço de IP 192.168.0.15 e porta 5005, completando, deste modo, o passo 4 do diagrama. Esses procedimentos podem ser observados na Figura 4.6, onde é possível notar que há uma duplicação das mensagens, nas quais somente é alterado o endereço IP de origem ou de destino.

Assim, as aplicações cliente e servidor TCP não precisam saber como funciona o mecanismo de redirecionamento que ocorre no roteador. Ele é feito normalmente, apenas substituindo o endereço de origem e de destino quando necessário.

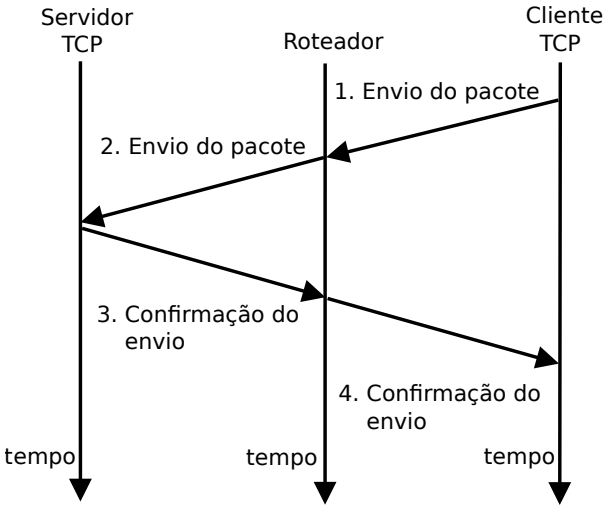

Figura 4.7: Diagrama ilustrando envio e confirmação de um pacote TCP entre o cliente e o servidor TCP.

4.3 Resumo conclusivo

Após o estudo dos ambientes e da verificação da corretude do cliente e do servidor neste capítulo, foi possível então passar para uma próxima etapa de testes, empregando, agora, uma simulação de um ambiente real, como será descrito no Capítulo 5. Pôde-se perceber que o cliente transmite uma mensagem correta para o servidor, contendo todos os campos do pacote conforme esperados e descritos na Seção 3.3, sob o ambiente mostrado na Figura 4.1. Desta maneira, é possível verificar que o protocolo PCP funciona corretamente conforme estudado e avaliado, como uma solução válida para travessia de NAT.

Capítulo 5

Avaliação de aplicações sobre o PCP

Para analisar o PCP como uma solução válida para aplicações em um cenário mais realístico, foi utilizado, neste capítulo, dois *softwares* de *backup*: o Bacula e o Rsync. Eles foram utilizados em cenários onde, atualmente, não funcionariam sem o auxílio de uma ferramenta de travessia de NAT, pois sem essa ferramenta não é possível que um agente externo se comunique com um dispositivo atrás de uma rede com NAT.

Como citado em [45], um *backup* pode ser descrito como uma réplica de qualquer dado que pode ser utilizada para recuperar sua forma original. Logo, um *software* de *backup* deve oferecer recursos importantes para auxiliar nisso, como: agendamento de *backup* para automatizar o processo; compressão de arquivos, para economizar espaço em disco; criptografia, para guardar os arquivos com mais segurança. Na maioria dos sistemas é importante a realização de *backups* periódicos, pois esta é uma das únicas formas de recuperação de informações em casos de pane. Além disso, é importante que as ferramentas de *backup* possam ser configuradas em diferentes ambientes, como os que serão vistos neste capítulo.

Como dito anteriormente, utilizar-se-ão duas ferramentas de *backup* em diferentes cenários para mostrar o PCP como solução válida em travessias de NAT. Elas foram escolhidas por serem bem conhecidas entre os *softwares* livres de *backup*, além de não cobrirem o cenário onde um servidor e um cliente se encontram em redes diferentes sem o mapeamento de endereço e porta do cliente para a rede externa, o que pode ser feito com o uso do PCP.

Neste capítulo serão avaliados dois cenários de testes, observando o tempo que cada uma das operações dos *softwares* de *backups* levarão, analisando seu desempenho e a velocidade de transmissão de dados, no primeiro deles realizar-se-ão cinco testes sobre um ambiente sem o uso do PCP, que irão mensurar o tempo decorrido e a vazão dos bits por segundo. Já no segundo teste, será utilizado um ambiente onde há um cliente e um servidor em redes distintas, utilizando o Protocolo de Controle de Portas para fazer

a travessia. Neste cenário será mensurado também a vazão dos bits e a quantidade de tempo que será tomado para que sejam realizados os testes.

5.1 Aplicações que podem utilizar o PCP

5.1.1 Bacula

A escolha desse programa foi baseada em pesquisas feitas acerca de *softwares* gratuitos e de código aberto de *backup* mais utilizados no mundo para uso em empresas e no uso pessoal [46], [47] e [48]. E, assim, nota-se que o Bacula está entre os primeiros colocados do *ranking* dos mais usados.

O Bacula é um conjunto de *softwares* que permitem ao administrador do sistema a realização de *backups*, recuperação e a verificação de dados através de uma rede de computadores entre cliente e servidor. Mas, para que isso seja possível, são necessários alguns componentes essenciais para a sua composição, como o serviço do diretor do Bacula, o console do Bacula, o serviço de arquivos Bacula, o serviço de armazenamento Bacula, o catálogo e o monitor do Bacula, tendo cada componente sua tarefa específica [49].

O Bacula é um sistema sofisticado de *backup* e foi feito para ser possível o gerenciamento de várias máquinas que desejam utilizar seus serviços. De maneira geral, o Bacula funciona com um servidor e com um ou mais clientes. Para executar um *backup* de algum arquivo de algum cliente, é necessário editar as configurações do Bacula no servidor. Logo, um cliente não consegue pedir um *backup* ao servidor, sem este saber o que está acontecendo através dos arquivos de configuração.

Na Figura 5.1, o servidor cria os arquivos de configuração, onde ficam todas as informações de clientes, de portas, do servidor propriamente dito, do *storage*, de *jobs*, de tempo de *backups* e *restores*, entre outras funcionalidades. Ao criar os arquivos de configuração básicos, é criado um *storage* no próprio servidor, configurando para se conectar na porta 9103, padrão.

Já na Figura 5.2, o servidor cria os arquivos de configuração, onde contém todas as informações do cliente, tal como seu IP, a porta padrão de conexão, a rotina de *backups* e *restores*, entre outras funcionalidades. E o cliente, por sua vez, configura através de seus arquivos de configuração o IP do servidor, seu nome e a sua senha de conexão.

Neste trabalho, o Bacula realizará um *backup* localmente no servidor, apesar de ser perfeitamente possível configurar os *backups* para serem armazenados em outra máquina. Este *backup* pode ser feito através de um cliente na mesma máquina do servidor ou através de um *host* na rede interna ou externa. O servidor se conecta ao *storage*, armazenando o conteúdo no IP onde está inserido nas configurações.

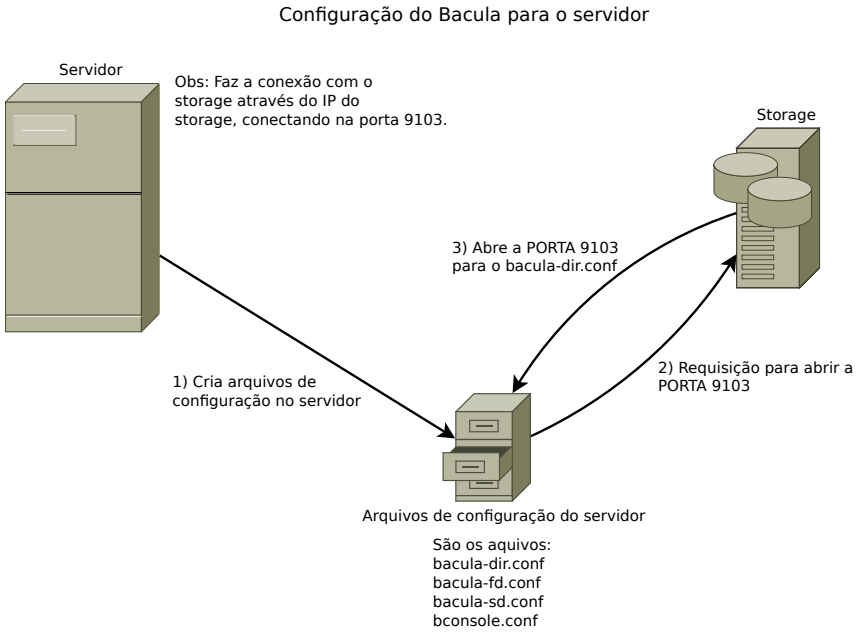

Figura 5.1: Topologia do servidor.

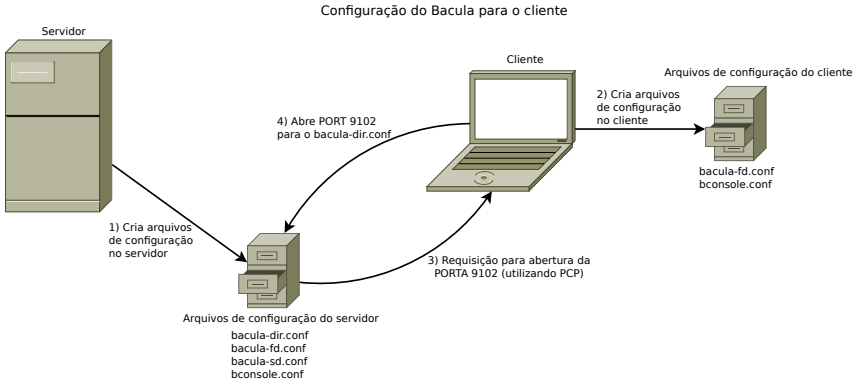

Figura 5.2: Topologia do cliente.

5.1.2 Rsync

O Rsync é uma ferramenta de *backup* mais simples que o Bacula. De acordo com a sua documentação, que se encontra em [50], o Rsync é uma ferramenta de cópia de arquivos rápida e extraordinariamente versátil. Pode copiar localmente, ou entre dois *hosts* através de um *remote shell*, que é uma ferramenta de acesso remoto. Ele oferece um grande número de opções que controlam cada aspecto de seu comportamento e permite uma especificação muito flexível do conjunto de arquivos a serem copiados. É famosa por seu algoritmo de transferência delta, que reduz a quantidade de dados enviados pela rede, enviando apenas as diferenças entre os arquivos de origem e os arquivos existentes no destino.

Baseado no comando de cópia segura, o Protocolo de Cópia Segura (do inglês, *Secure*

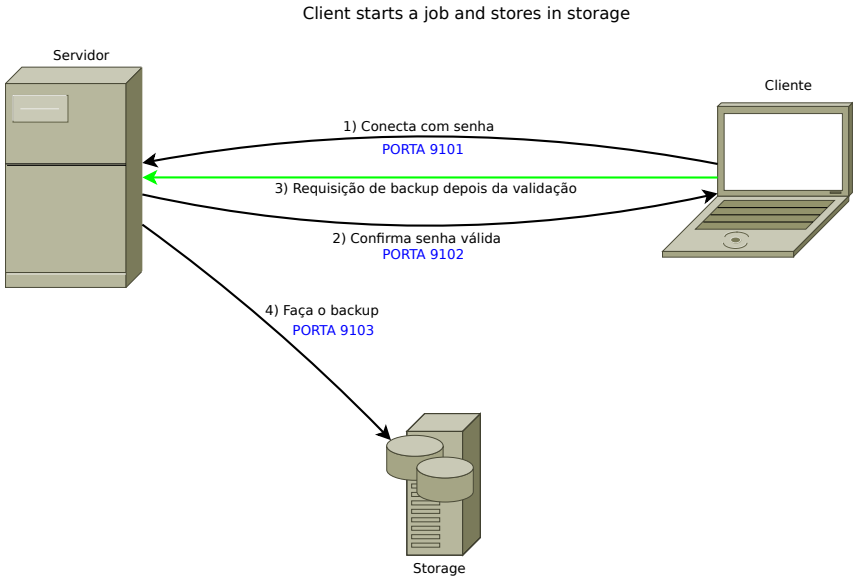

Figura 5.3: Topologia para *backup*.

Copy Protocol - SCP), o Rsync também utiliza o protocolo *Secure Shell* (SSH) para fazer a sincronização segura de dados entre duas máquinas. Para isso é necessário que a máquina que age como servidor esteja executando o serviço do Rsync em plano de fundo e que o cliente e o servidor estejam na mesma rede. Caso estejam em redes diferentes é necessário abrir a porta ssh para uso externo.

Esta ferramenta foi criada com bastante foco em eficiência. A explicação da arquitetura do Rsync, assim como suas inspirações, pode ser vista em [51].

5.2 Problema

Em um cenário onde um servidor Bacula ou Rsync executa em uma rede diferente da utilizada pelo cliente, não é possível, ao servidor, iniciar um *backup* dos arquivos do cliente, pois qualquer comunicação onde o agente externo ao NAT queira iniciar a troca de mensagens com um dispositivo atrás dele, essa conversa será "barrada" pelo NAT. Assim, a solução proposta neste trabalho envolve o uso do PCP para fazer o mapeamento das portas utilizadas pelo cliente Bacula e Rsync no roteador e, deste modo, realizar a comunicação bem sucedida entre eles.

Para a realização dos experimentos com o Bacula e o Rsync, foram configurados dois ambientes de testes. O primeiro deles consiste em apenas duas máquinas: uma representando o cliente e outra o servidor. Já o segundo, envolve, além do cliente e do servidor, o uso de um roteador com NAT. Em ambos os contextos, o objetivo é iniciar um *backup* de

algum arquivo do cliente a partir do servidor. A seguir, há uma descrição mais detalhada de cada cenário.

5.3 Cenários de avaliação

Os cenários descritos abaixo são formulados de uma maneira simples, mas que fosse possível realizar os testes necessários, realizando uma verificação para uma validação correta do que foi alvitrado. Sendo assim, o cenário 1 proposto foi-se utilizado o cliente e o servidor na mesma rede local e, consecutivamente, o cenário 2 descreve a situação em que o cliente e o servidor estão dispostos em redes diferentes. Foram utilizados dois cenários de avaliação para que fosse possível a verificação do uso dos *softwares* de *backup*. Eles atuarão como comparativo dos casos em que suas operações fossem realizadas em um ambiente sem a utilização do protocolo PCP em sua implementação e em um outro ambiente com a utilização dele.

5.3.1 Cenário 1

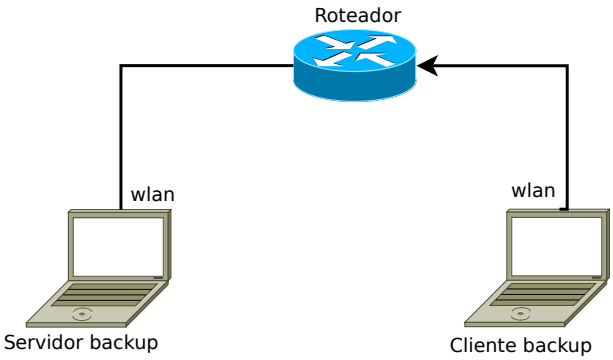

Figura 5.4: Esquema do cenário 1.

Neste cenário foram realizados dois testes, em que o primeiro deles o servidor utilizado é simulado por uma máquina virtual, utilizando o software VirtualBox, versão 5.1.34, com o sistema operacional Ubuntu, versão 16.04.4 LTS, que executa o Bacula versão 9.0.6. E o cliente, por sua vez, utilizou uma máquina com o sistema operacional Ubuntu versão 19.04, executando o Bacula versão 9.4.3.

No segundo teste, foram utilizados o Rsync versão 3.1.2 tanto no servidor, quanto no cliente, utilizando os mesmos computadores do teste anterior.

Em ambos os testes deste cenário, o servidor e o cliente estão conectados na mesma rede local, como esquematizado na Figura 5.4, por Wi-Fi. Este é um cenário coberto

atualmente pelo Bacula e pelo Rsync e foi reproduzido a fins de comparação com o mundo real.

5.3.2 Cenário 2

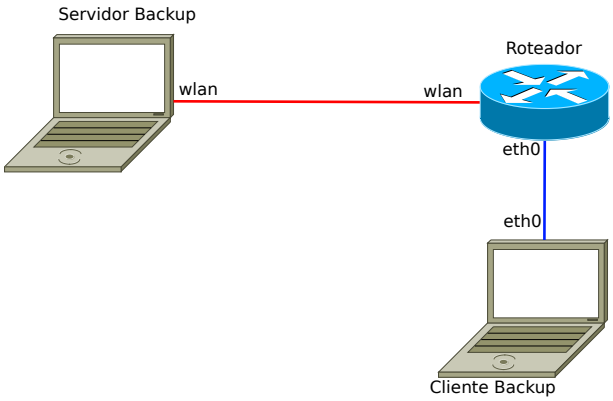

Figura 5.5: Esquema do cenário 2.

Já neste cenário, também foram realizados dois testes, com as mesmas configurações dos dispositivos utilizados no cenário 1, com o servidor utilizando uma máquina virtual e o cliente utilizando o sistema operacional Ubuntu.

Os dois testes foram montados de maneira semelhante ao ambiente descrito no Capítulo 4, que pode ser visto na Figura 4.1. No qual, o servidor e o cliente utilizados são os mesmos. Aqui, o cliente se encontra atrás do NAT, e é representado pelo Cliente Backup, enquanto que o servidor é representado pelo Servidor Backup, em uma rede externa à rede ao cliente. Na Figura 5.5, há o esquema deste cenário.

A diferença dos dois testes realizados é que no primeiro deles, ambos o cliente e o servidor utilizam o Bacula como a solução de *backup*. Já no segundo teste, será utilizado o Rsync como o software livre de *backup*, para comparação com o Bacula.

5.3.3 Teste de largura de banda

O objetivo deste teste é identificar qual a largura de banda média da rede em cada cenário para compará-los e, também, para saber o quanto desta largura as ferramentas de *backup* conseguem utilizar. A largura de banda média foi mensurada utilizando o *software* gratuito IPerf, que detecta, através de uma conexão TCP ou UDP, qual o valor máximo da velocidade de transmissão de dados naquele momento. Nos testes a seguir, apresentados na Tabela 5.1, serão utilizadas conexões TCP para mensurar os valores almejados, pois o Rsync não possui suporte à conexões UDP [52]. Mas, para uma comparação da velocidade

medida por ele, foram-se obtidos também valores através da ferramenta incluída no próprio Bacula, que também faz essa aferição, medindo a velocidade média em que o Bacula consegue escrever e ler dados do cliente. A tabela a seguir apresenta a média e o desvio padrão de cinco testes diferentes de larguras de banda medidas nos dois cenários descritos nas Seções 5.3.1 e 5.3.2.

Tabela 5.1: Velocidade média de transmissão de dados x cenários.

Cenário	IPerf (Mbits/s)	Bacula (Mbits/s)
Cenário 1	$20{,}13 \pm 5{,}07$	$21{,}18 \pm 3{,}91$
Cenário 2	$23{,}63 \pm 3{,}82$	$23{,}13 \pm 3{,}32$

Através da Tabela 5.1, pôde ser analisado que os valores apanhados para os testes foram semelhantes, havendo uma diferença de 1,05 Mbits/s para o Cenário 1 e de 0,5 Mbits/s para o Cenário 2, mostrando-se valores aproximados, utilizando, desta maneira, os valores obtidos no IPerf como base para a realização dos testes. O desvio padrão obtido dos testes foi consideravelmente alto por conta da oscilação da rede utilizada, de modo que, somente ao serem executados dois testes em um mesmo minuto, os valores já se diferenciavam em 12 Mbits/s.

5.3.4 Dados dos testes

Nos dois cenários, o objetivo é iniciar um *backup* por parte do servidor, tanto no Bacula, quanto no Rsync. Ao final da execução de um *job*, seja *backup* ou *restore*, é possível visualizar algumas estatísticas interessantes, como o tempo total para realizar o *job*, a velocidade média da transmissão dos dados e a taxa de compressão dos arquivos.

Nos experimentos foram utilizados dois arquivos de 500MB: um com alta capacidade de compactação (36.3% de compressão com o gzip), sendo indicado por Arquivo1 e um segundo com nenhuma capacidade de compactação (0% de compressão com o gzip), sendo nomeado por Arquivo2. O tipo de *backup* feito nas duas ferramentas foi o completo, com velocidade de transferência ilimitada. Destarte, as Tabelas 5.2 e 5.3 apresentam os resultados obtidos comparando os tempos tomados para fazer o *backup* e o *restore* em cada cenário com cada uma das ferramentas. Os experimentos foram realizados cinco vezes, e seus resultados foram computados calculando a média e desvio padrão de todos esses testes.

Através dos resultados alcançados, conforme mostrados nas tabelas acimas, foi possível ver que ambos os tempos de *backup* e *restore* correspondentes aos dois arquivos foi superior no Bacula, quando comparado ao Rsync. Para o Cenário 1, com o Bacula, o tempo total de *backup* e *restore* foi de 307,99 segundos para o Arquivo1 e de 467,34 segundos para o

Tabela 5.2: Tempo em segundos para cada operação nos dois cenários utilizando o Bacula.

Cenários	Backup		Restore	
	Arquivo 1	Arquivo 2	Arquivo 1	Arquivo 2
Cenário 1	$117{,}33 \pm 35{,}26$	$203{,}67 \pm 23{,}04$	$190{,}66 \pm 62{,}77$	$263{,}67 \pm 10{,}40$
Cenário 2	$130{,}67 \pm 30{,}00$	$186{,}33 \pm 44{,}04$	$169{,}67 \pm 66{,}14$	$228{,}66 \pm 66{,}94$

Tabela 5.3: Tempo em segundos para cada operação nos dois cenários utilizando o Rsync.

Cenários	Backup		Restore	
	Arquivo 1	Arquivo 2	Arquivo 1	Arquivo 2
Cenário 1	$102{,}33 \pm 16{,}54$	$215{,}66 \pm 22{,}69$	$136{,}33 \pm 17{,}75$	$247{,}00 \pm 64{,}27$
Cenário 2	$108{,}33 \pm 09{,}10$	$184{,}00 \pm 22{,}11$	$128{,}00 \pm 12{,}36$	$199{,}33 \pm 47{,}99$

Arquivo2. Para o Rsync, o tempo total para o Arquivo1 foi de 238,66 segundos e para o Arquivo2 foi de 482,66. Assim, pode-se notar que o tempo total para o Arquivo1 do Bacula foi superior ao do Rsync em 69,33 segundos. Já, para o Arquivo2, o tempo do Bacula foi superior em apenas 4,68 segundos, com este arquivo tendo 0% de compressão.

Para o Cenário 2, no qual é aplicado o protocolo PCP no cliente e no servidor, o Bacula obteve um tempo de 300,34 segundos para o Arquivo1, sendo que para o Rsync o tempo total foi de 236,33 segundos. Para o Arquivo2, o Bacula obteve uma quantidade de 414,99 segundos e o Rsync de 383,33 segundos. Sendo assim, o Rsync foi mais rápido que o Bacula em 64,01 segundos para o Arquivo1. E, para o Arquivo2, ele foi mais rápido em 31,66 segundos, mostrando-se mais veloz que tanto em dispositivos na mesma rede, quanto que para dispositivos em rede distintas, com o cliente atrás do NAT.

Desta forma, para ambos os cenários, o Bacula precisa de um tempo maior para realizar as operações de *backup* e *restore*, visto que o seu algoritmo de compressão demanda mais tempo que o Rsync, como foi visto pelos resultados. Uma maneira de tentar resolver esse problema é utilizar o formato de compressão LZO, que demanda menos recursos que o algoritmo de GZIP do Bacula [53], pois a compressão é realizada no cliente, utilizando processamento do cliente [54].

Para as operações de *backup* do Arquivo1 foram gerados dois gráficos comparativos dos cenários descritos, um utilizando o Bacula e outro utilizando o Rsync para *backup*. Nos gráficos, a cor vermelha indica o Cenário 2, enquanto que a cor verde indica o Cenário 1. Dois deles serão apresentados a seguir, através das Figuras 5.6 e 5.7.

Através dos gráficos, pôde ser notado que a velocidade de transmissão do Rsync é superior ao Bacula em vários momentos, ultrapassando a marca de 4800 *bits* por segundo, para o Cenário 1 e passando 4200 *bits* por segundo para o Cenário 2. Nestes cenários

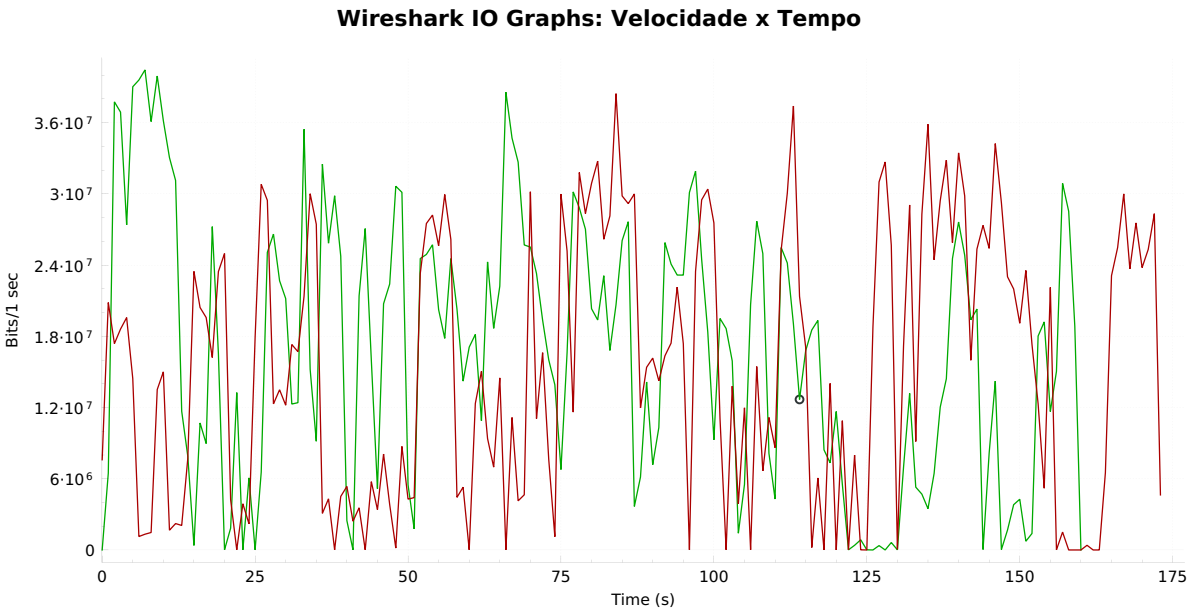

Figura 5.6: Velocidade de transmissão de bits por segundo de um *backup* do Arquivo1 feito pelo Bacula.

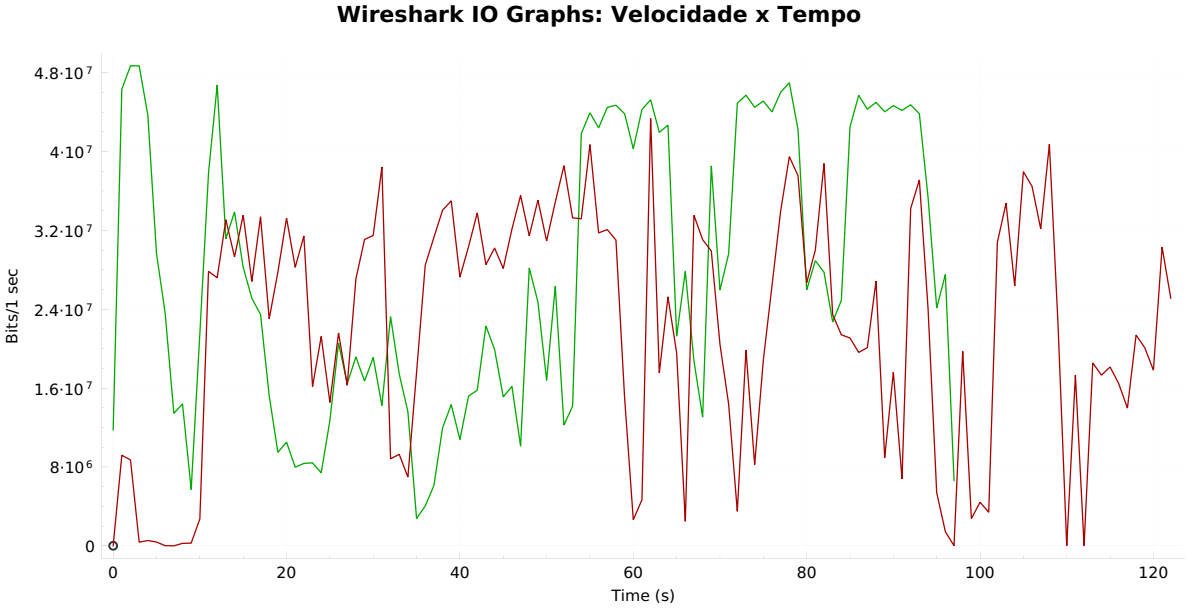

Figura 5.7: Velocidade de transmissão de bits por segundo em um *backup* do Arquivo1 feito pelo Rsync.

observa-se que a maioria dos *bits* sendo transmitidos foi maior que 3200 por segundo.

Já o Bacula, a maior marca que ele atingiu foi extrapassando a marca de pouco mais de 4000 *bits* por segundo para o Cenário 1, em contrapartida ao Cenário 2, que alcançou

aproximadamente 3800 *bits* por segundo no seu pico mais alto. Mas, apesar dele alcançar esse máximo, a transmissão do Cenário 1 ficou em sua maioria na média em torno de 2400 *bits* por segundo. O Cenário 2 ficou com uma média por volta de 2200 *bits* por segundo. Por sua vez, o Rsync ficou com uma média de aproximadamente 3200 *bits* por segundo para o Cenário 1 e com uma média perto de 3000 *bits* por segundo para o Cenário 2.

5.4 Resumo conclusivo

É possível notar que ambas velocidades médias e velocidades máximas alcançadas pelo Rsync é superior ao Bacula, influenciando também em uma menor quantidade de tempo para serem executadas as tarefas necessárias. Mas, conquanto o Bacula apresente um menor desempenho em relação ao Rsync, ele apresenta vários outros benefícios em relação a ele, como por exemplo a definição de diversos *jobs* para um determinado cliente, possibilitando que seja realizado um *backup* mais automatizado e mais dinâmico de forma mais simples. Além disso, o Bacula permite que vários clientes sejam configurados ao mesmo tempo, ao contrário do Rsync, que só permite que um cliente faça uma requisição por vez, sem o uso de *softwares* de terceiros. Uma outra vantagem do Bacula em relação ao Rsync é a possibilidade da escolha do arquivo de compactação que o usuário deseja utilizar, sendo possível escolher formatos que beneficiem transmissões mais rápidas ou uma maior compressão dos arquivos. Outro ponto positivo que o Bacula possibilita é a consulta do histórico de operações que foram realizadas, proporcionando uma maior segurança ao usuário, por saber quais foram as vezes que foram requisitados os *backups* ou *restores* e uma maior manutenção dos processos. Por contrapartida, o Rsync é mais simples de ser utilizado para tarefas comuns, como *backups* não automatizados e operações de requisições de operações únicas (restauração ou armazenamento de dados). Ademais, o Rsync é mais eficiente, como foi mostrado nos testes anteriores, possuindo um tempo de *backup* e *restore* menores que o Bacula. Apesar de todas as vantagens e desvantagens, o Bacula pode ser considerado recomendado para uso em empresas ou caso deseje-se fazer operações para um grupo maior de pessoas. Já o Rsync é recomendado para o uso pessoal, por ser mais simples e por possuir uma maior facilidade em ser utilizado.

Ao deixar o PCP responsável pelo mapeamento de portas, as aplicações de *backup* funcionam sem precisarem entender o que está acontecendo por trás do processo de tradução de endereços. Desta forma, não é necessário "saber" que o cliente final não é o roteador, precisando apenas do necessário para acessar o cliente, que, no caso, é um endereço de IP/Porta que apontam para o roteador. Logo, o PCP cumpre bem o seu papel nestes casos, permitindo que os servidores consigam iniciar um pedido de *backup* ou *restore* no

cliente, sem que haja uma influência direta do PCP no fluxo de dados da rede que está sendo utilizada.

Capítulo 6

Conclusão

O NAT surgiu graças ao grande uso do IPv4, o que levou à exposição de uma de suas maiores limitações, o número de endereços de IP disponíveis para serem utilizados. Como visto neste trabalho, ele é uma boa solução para esta limitação, mas também deve-se notar que ele traz consigo um grande problema relacionado a como estabelecer uma comunicação entre dois *hosts* que estão em redes diferentes.

Como visto extensamente, existem vários protocolos que buscam resolver o problema de travessia em NATs. Este trabalho propôs resolver este problema com o uso do PCP, por ser um protocolo simples e por ser *peer-to-peer*, isto é, que não necessita de uma máquina auxiliar para ajudar na solução. Para seu funcionamento correto, ele depende apenas que dispostivos NAT, como roteadores, implementem o servidor PCP.

No trabalho, também foi proposto mostrar um lado mais prático do PCP, aplicando ele na prática com ferramentas que se beneficiam de seu uso, como os programas de *backup*. O uso de uma ferramenta de travessia de NAT complementa muito bem as ferramentas escolhidas, aumentando o escopo de ação delas. Desta forma, a utilização do PCP foi mostrada completamente viável em um ambiente mais restrito, em comparação com os NATs de grande escala, onde ele é mais utilizado.

Com a implementação do PCP foi possível analizar também quais foram as vantagens e desvantagens de um *software* de *backup* em relação ao outro. De modo que o Bacula seja mais recomendado por ser mais robusto e mais automatizado para a implementação em meios empresariais, ao contrário do Rsync, que é mais recomendado para o uso pessoal ou para um uso mais simplificado para quem busca essa solução de armazenamento de seus dados.

6.1 Trabalhos futuros

Em termos de aplicação do PCP, um próximo passo seria implementar o PCP diretamente nas ferramentas utilizadas no trabalho, pois aqui todo o mecanismo de fazer o mapeamento e fazer a publicação do par IP:Porta, do cliente para o servidor, foi realizado manualmente. Ao implementar o PCP em comunhão com estas aplicações, o processo seria automático e transparente para o usuário.

Além disso, com um foco maior no PCP, seria possível aprofundar mais a implementação do protótipo utilizado no trabalho, visando cobrir mais o que é especificado em sua RFC, como por exemplo os mecanismos de retransmissão.

Referências

[1] Stats, Internet Live: *Total number of Websites.* Disponível em https://www.internetlivestats.com/total-number-of-websites/#trend., 2018. Acesso em Julho de 2019. xii, 3

[2] TANENBAUM e Andrew S.: *Redes de Computadores.* Editora Campus, Rio de Janeiro, fourth edição, 2003. xii, 3, 7, 8, 9, 10, 11

[3] Agency, Defense Advanced Research Projects: *INTERNET PROTOCOL.* RFC 791, sep 1981. xii, 11

[4] KUROSE e ROSS: *Redes de computadores e a internet: uma abordagem top-down.* PEARSON, sixth edição, 2014. xii, 7, 8, 9, 11, 20, 66

[5] Wing, D., S. Cheshire, M. Boucadair, R. Penno e P. Selkirk: *Port Control Protocol (PCP).* RFC 6887, apr 2013. xii, 24, 25, 26, 27, 28, 29, 30, 32, 42, 43, 45

[6] Mahy, R., P. Matthews e J. Rosenberg: *Trasversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN).* RFC 5766, apr 2010. xii, 16, 35, 36, 37, 39, 40

[7] Faria, Henrique: *Diagrama Opções do Director (bacula-dir.conf) – DEPRECADO.* http://www.bacula.com.br/diagrama-opcoes-do-director-bacula-dir-conf/. Acesso em Março de 2019. xiii, 81

[8] Navarro, R. F.: *A Evolução dos Materiais. Parte1: da Pré-história ao Início da Era Moderna.* Revista Eletrônica de Materiais e Processos, v.1, 1 (2006) 01-11 ISSN 1809-8797, páginas 2–3, 2006. 1

[9] Souza, Edwin Uisy de: *O transporte público urbano como suporte para o desenvolvimento do turismo em Natal/RN.* Tese de Mestrado, Universidade Federal do Rio Grande do Norte, 2013. 1

[10] Saito, José Hiroki e O ábaco foi inventado na Babilônia: *Resumo sobre a História da Computação.* Augusta, 1842:43, 2005. 1

[11] Pugh, Emerson W.: *Building IBM: Shaping an Industry and Its Technology.* MIT Press, 1995. 1

[12] Sheriguda, Ibrahimpatnam, Reddy e Ranga: *The Technology Journey from Floppy disk to Cloud storage.* International Journal of Computer Technology and Applications, 2:446–450, maio 2011. 2

[13] Wolpin, Stewart: *DVD Takes The Day*. Invention & Technology, 24, 2010. 2

[14] Insights, CB: *The Future Of Data Centers*. https://www.cbinsights.com/research/future-of-data-centers/, jan 2019. 2

[15] Lins, Bernardo Felipe Estellita: *A evolução da Internet: uma perspectiva histórica*. Cadernos Aslegis, 17(48):11–45, 2013. 2

[16] Yurin, Maxim: *THE HISTORY OF BACKUP*. www.backuphistory.com. Acesso em Junho de 2019. 2

[17] TANENBAUM, ANDREW S. e DAVID J. WETHERALL: *Computer networks*. Pearson, fifth edição, 2011. 3, 7, 67

[18] Srisuresh, P. e M. Holdrege: *IP Network Address Translator (NAT) Terminology and Considerations*. RFC 2663, aug 1999. 3, 14, 15

[19] Kang, Sung Woon: *System and method for network address translation and session management*, aug 2004. 4

[20] Filippetti, Marcos A.: *CCNA 3.0 – Guia Completo de Estudo*. Visual Books, Florianópolis, 2002. 6

[21] Soares, Luiz Fernando Gomes, Guido Lemos e Sérgio Colcher: *Redes de computadores. Das LANs, MANs e WANs às Redes ATM*. Editora Campus, second edição, 1995. 7

[22] Bora, Gaurav, Saurabh Bora, Shivendra Singh e Sheikh Mohamad Arsalan: *OSI Reference Model: An Overview*, volume 7. International Journal of Computer Trends and Technology (IJCTT), Florianópolis, fourth edição, jan 2014. 9

[23] A, GALLO M. e HANCOCK W. M.: *Comunicação entre Computadores e Tecnologias de Rede*. Pioneira Thomson Learning, 2004. 9

[24] Group, Network Working: *Address Allocation for Private Internets*. RFC 1918, feb 1996. 11, 23

[25] Cotton, M. e L. Vegoda: *Special Use IPv4 Addresses*. RFC 5735, jan 2010. 11

[26] Droms, R.: *Dynamic Host Configuration Protocol*. RFC 2131, mar 1997. 12

[27] Stegel, Tine, Janez Sterle, Janez Bešter e Andrej Kos: *SCTP association between multi-homed endpoints over NAT using NSLP*. Univerza v Ljubljani, Fakulteta za elektrotehniko, 2008. 16

[28] Networks, Juniper: *Port Control Protocol Overview*. https://www.juniper.net/documentation/en_US/junos/topics/concept/nat-port-control-protocol.html, jul 2018. Acesso em Julho de 2019. 16

[29] Srisuresh, P., Kazeon Systems, B. Ford, M.I.T e D. Kegel: *State of Peer-to-Peer (P2P) Communication across Network Address Translators (NATs)*. RFC 5128, mar 2008. 16, 67

[30] Ford, Bryan, Pyda Srisuresh e Dan Kegel: *Peer-to-Peer Communication Across Network Address Translators*. CoRR, abs/cs/0603074, 2006. 17, 18, 19

[31] Widmer, Christopher Daniel: *NAT Traversal Techniques and UDP Keep-Alive Interval Optimization*. Tese de Mestrado, Florida Institute of Technology, Melbourne, Florida, jul 2015. 19

[32] Hoepers, Cristine e CEPTRO.br: *Transição IPv4 -> IPv6: Desafios e Riscos*. ICCYBER, sep 2012. `https://www.cert.br/docs/palestras/certbr-ceptrobr-iccyber2012.pdf`, Acesso em Agosto de 2019. 19

[33] Tacio, Paulo: *[TOP 5] OS MELHORES SNIFFERS GRATUITOS*. http://www.mundodoshackers.com.br/top-5-os-melhores-sniffers-gratuitos, feb 2011. Acesso em Julho de 2019. 20

[34] Foundation, Wireshark: *About Wireshark*. https://www.wireshark.org/index.html#-aboutWS. Acesso em Julho de 2019. 20

[35] Wu, Chwan Hwa (John) e J. David Irwin: *Introduction to Computer Networks and Cybersecurity*. CRC Press - Taylor & Francis Group, London, New York, 2013. 21

[36] Cheshire, S. e M. Krochmal: *NAT Port Mapping Protocol (NAT-PMP)*. RFC 6886, apr 2013. 21, 22, 24

[37] Rosenberg, J., R. Mahy, P. Matthews e D. Wing: *Session Traversal Utilities for NAT (STUN)*. RFC 5389, oct 2008. 32, 33, 34, 35, 39

[38] Rosenberg, J., J. Weinberger, C. Huitema e R. Mahy: *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. RFC 3489, mar 2003. 32

[39] Irvine, David: *DHT-based NAT Traversal*. MaidSafe, página 1, sep 2010. Disponível em https://docs.maidsafe.net/Whitepapers/pdf/DHTbasedNATTraversal.pdf. 39

[40] Mäenpää, Jouni: *Framework Architecture for Decentralized Communications*. Tese de Doutoramento, Aalto University, 2013. Disponível em https://pdfs.semanticscholar.org/e692/1eaf40a84bb24dfb547885cd2502b89f126a.pdf. 40

[41] Kantola, R.: *Lecture Notes in Signaling Protocols - Network Address Translators and NAT Traversal*. Computer Network Laboratory of Helsinki University of Technology, jan 2015. 40

[42] Bernard, Thomas: *Miniupnp*. http://miniupnp.free.fr/. Acesso em Maio de 2019. 42, 44

[43] Tatrai, Peter: *PCP*. https://github.com/libpcp/pcp/. Acesso em Janeiro de 2019. 42

[44] Foundation, Open Connectivity: *UPNP STANDARDS & ARCHITECTURE*. https://openconnectivity.org/developer/specifications/upnp-resources/upnp, 2019. Acesso em Julho de 2019. 44

[45] Guise, De e Preston: *Enterprise systems backup and recovery: a corporate insurance policy*. Auerbach Publications, 2008. 49

[46] Vijayakumar, Gobind: *The Must-Have Open Source Tools for Backup and Recovery*. http://opensourceforu.com/2015/10/ten-must-have-open-source-tools-for-backup-and-recovery, 2015. Acesso em Junho de 2019. 50

[47] Harvey, Cynthia: *Open Source Storage: 49 Tools for Backup and Recovery*. https://www.enterprisestorageforum.com/backup-recovery/open-source-storage-49-tools-for-backup-and-recovery.html, 2014. Acesso em Junho de 2019. 50

[48] SourceForge: *Bacula*. http://sourceforge.net/projects/bacula. Acesso em Junho de 2019. 50

[49] Sibbald, Kern: *What is Bacula*. http://www.bacula.org/9.0.x-manuals/en/main/What_is_Bacula.html. Acesso em Maio de 2018. 50

[50] *rsync(1) Linux User's Manual*, July 2013. 51

[51] Tridgell e Andrew: *Efficient algorithms for sorting and synchronization*. Tese de Doutoramento, The Australian National University, feb 1999. 52

[52] Tridgell, Andrew, Paul Mackerras e Wayne Davison: *rsync*. https://download.samba.org/pub/rsync/rsync.html, jan 2018. Disponível sobre GNU General Public License. Acesso em agosto de 2019. 54

[53] Sibbald, Kern: *New Features in 5.2.x*. https://www.bacula.org/5.2.x-manuals/en/main/main/New_Features_in_5_2_x.html, 2013. Acesso em Agosto de 2019. 56

[54] Ksiaskiewcz, Allan Patrick: *Compressão*. https://groups.google.com/forum/#!topic/bacula-backup-pt-br/83taS1ECtLs, 2016. Acesso em Agosto de 2019. 56

[55] Audet, F. e C. Jennings: *Network Address Translation (NAT) Behavioral Requirements for Unicast UDP*. RFC 5128, jan 2007. 67, 68

[56] Hertzorg, Raphäel e Roland Mas: *The Debian Administrator's Handbook*. Freexian SARL, 2015. 69

[57] Martinez, Erick: *Bacula - Parte 1 - Instalação e configuração do servidor de backup*. https://www.mundotibrasil.com.br/bacula-parte-1-instalacao-e-configuracao-do-servidor-de-backup/. Acesso em Fevereiro de 2018. 80

[58] Faria, Henrique: *Comandos de Compilação e Instalação Bacula 9.0.6*. http://www.bacula.com.br/comandos-de-compilacao/comment-page-1/. Acesso em Março de 2018. 80

[59] Anicas, Mitchell: *How to Install Bacula Server on Ubuntu 14.04*. https://www.digitalocean.com/community/tutorials/how-to-install-bacula-server-on-ubuntu-14-04. Acesso em Março de 2018. 80

Glossário

Daemon

Um *Deamon* é um *software* que executa algum processo em plano de fundo. Ele possui o propósito de manipular processos requisitados que um sistema necessite, ou seja, ele roda em plano de fundo e dá um resultado, ou responde, aos serviços que o chamaram.

Host

Dispositivos computacionais interligados pela Internet. São definidos como *hosts* (hospedeiros) pois hospedam programas de aplicação [4].

Computador que acessa recursos compartilhados nos servidores em uma rede local. Qualquer computador em uma rede pode ser servidor e/ou cliente, basta que ele disponibilize e/ou acesse recursos [4].

Middlebox

Um dispositivo que pode manipular o tráfego para uma determinada funcionalidade, como por exemplo o *firewall*, que verifica tráfegos maliciosos.

Comutação de pacotes

É o envio de uma mensagem de um sistema de origem a um sistema de destino, passando por enlaces de comunicação e comutadores de pacotes, que são dispositivos que fazem o transporte de pacotes de uma origem a um destino, como por exemplo roteadores [4].

Comunicação fim a fim

É um caminho entre o transmissor e o receptor em uma comunicação. Por exemplo na comutação de circuitos, em que, ao realizar uma chamada telefônica, o caminho entre os dois agentes fica ocupado enquanto a chamada não é finalizada [17].

Endpoint-Independent Mapping

De acordo com a RFC 4787 [55], EIM são definidos como o reuso do mapeamento de porta do NAT para os próximos pacotes que serão enviados do mesmo IP/Porta internos para qualquer endereço IP/Porta externo [55].

Endpoint-Independent Filtering

O envio de pacotes do lado interno do NAT para um destino externo é suficiente para permitir que qualquer pacote volte para o cliente que o enviou [29].

Endpoint-Dependent Mapping

É a combinação dos conceitos de ADM e APDM.

Endpoint-Dependent Filtering

É a combinação dos conceitos de ADF e APDF.

Address-Dependent Mapping

O NAT reusa o mapeamento de porta para os próximos pacotes enviados do mesmo endereço IP/Porta internos para o mesmo endereço IP externo, independentemente da porta externa [55].

Address and Port-Dependent Mapping

O NAT reusa o mapeamento de porta para os próximos pacotes enviados do mesmo endereço IP/Porta interno para o mesmo endereço IP/Porta externo, enquanto o mapeamento ainda está ativo [55].

Address-Dependent Filtering

O NAT filtra os pacotes não destinados ao endereço de IP/Porta internos. E também, o NAT filtrará os pacotes de um IP/Porta externos para um IP/Porta internos, caso o endereço IP/Porta interno ainda não tenha enviado pacotes para o IP externo, sendo a porta externa irrelevante [55].

Address and Port-Dependent Filtering

O Nat filtra os pacotes não destinados ao endereço de IP/Porta internos. E também, o NAT filtrará os pacotes de um IP/Porta externos para um IP/Porta internos, caso o endereço IP/Porta interno ainda não tenha enviado pacotes para o IP/Porta externo, sendo a porta externa relevante [55].

Anexo I

Instalação e configuração do roteador

Antes de começar a configuração do roteador do linux deve-se instalar alguns pacotes para o seu completo funcionamento. São eles

```
# apt-get install iptables-dev
```

Que contém as bibliotecas libipq, libiptc e libxtables, que provê uma Interface de Programação de Aplicativos (do inglês, *Application Programming Interface* - API) para a comunicação com ip_queue, usada para a comunicação com o netfilter e uma biblioteca chamada xtables necessária para o netfilter, consecutivamente.

```
# apt-get install bind9
```

Que faz a tradução de nomes de domínio para endereços IP. Ele será responsável por rodar o servidor DNS no roteador.

```
# apt-get install apparmor
```

"É um sistema de Controle de Acesso Mandatório construído sobre a interface de Módulos de Segurança Linux (do inglês, *Linux Security Modules* - LSM). Na prática, o kernel consulta o AppArmor antes de cada chamada do sistema para saber se o processo está autorizado a fazer a operação dada. Através desse mecanismo, o AppArmor confina programas a um conjunto limitado de recursos" [56].

```
# apt-get install isc-dhcp-server
```

Será esse pacote que implementará o servidor DHCP no roteador. Ele quem permitirá que seja feita a distribuição de IPs a cada cliente conectado.

Após a instalação dos pacotes, começa-se a configuração do roteador. Para isso, foi necessário criar regras NAT através do iptables, para que fosse possível a transmissão de

pacotes de uma interface de rede para outra. Neste trabalho, utilizou-se uma rede *wireless* para prover Internet para redes cabeadas. Sendo assim, todo pacote requerido por um ip interno do roteador criado foi passado através de uma rede cabeada, pela interface eth0 para uma rede *wireless*, utilizando a interface wlan0. Para que fossem feitas todas as devidas configurações, utilizou-se os seguintes passos:

1. Para a ativação do IP *forward* no *kernel* foi necessário mudar o valor de 0 para 1 nas configurações de *forwarding* do servidor da seguinte maneira:

   ```
   # echo 1 > /proc/sys/net/ipv4/ip_forward
   ```

2. Para conectar-se a rede cabeada à Internet foi necessário adicionar regras na tabela NAT da iptables, para os pacotes serem transmitidos à rede cabeada, no caso a eth0. No comando abaixo, o POSTROUTING corresponde ao pacote que está saindo imediatamente antes para a rede externa. Ele resultará que os pacotes da rede interna sejam devidamente corretos com um IP externo. O MASQUERADE fará o mascaramento do IP, criando um IP único para um dispositivo dentro da rede NAT. Para que isso se concretizasse, utilizou-se o seguinte comando:

   ```
   # iptables -t nat -A POSTROUTING -o wlan0 -j MASQUERADE
   ```

3. Para fazer o pacote percorrer da rede *wireless* para a rede interna precisa-se permitir que todos os pacotes de todos os clientes passem com sucesso, para isso utiliza-se o seguinte comando:

   ```
   # iptables -A FORWARD -i eth0 -j ACCEPT
   ```

4. Também é necessário que sejam feitas as configurações para os IPs da rede a ser roteada, como também da interface de rede que será utilizada como DHCP, no caso a wlan0. Sendo assim, edita-se o arquivo /etc/network/interfaces, adicionando as seguintes configurações:

   ```
   auto wlan0
   iface wlan0 inet dhcp

   auto eth0
   iface eth0 inet static
   ```

```
address 192.168.0.1
netmask 255.255.255.0
```

5. Deste modo, basta agora reiniciar os serviços de rede da seguinte maneira:

```
# /./etc/init.d/networking restart
```

6. Após realizado todos os passos anteriormente, precisamos configurar o servidor de DNS. Sendo assim, precisou-se do Bind9 no roteador, que permite que essa configuração possa ser feita. Depois de instalado o bind9, necessitou-se inserir manualmente o nome do servidor para que os clientes possam utilizá-lo:

```
# echo "nameserver 192.168.0.1">/etc/resolv.conf
```

7. Se o passo anterior não funcionar corretamente, ou seja, não for possível fazer um ping para google.com, deve-se alterar o arquivo /etc/bind/named.conf.options e adicionar o IP do seu roteador em *forwarders*, no caso do trabalho será o 10.0.0.1.

8. Agora, foi feita uma customização do roteador para que fosse possível guardar todos os registros de requisição de DNS. Para isso, adicionou-se as seguintes linhas no arquivo /etc/bind/named.conf.local:

```
logging {
   channel query.log {
      file "/home/pi/named/dns.log";
         severity debug 3;
            print-time yes;
   };
   category queries { query.log; };
};
```

Criar a pasta named e o arquivo dns.log. Ainda neste arquivo e dentro antes de adicionar o logging, adiciona-se uma zona local ao /etc/bind/named.conf.local, para um domínio que será chamado de “lan”: zone "lan" type master; file "/home/router/named/lan.db"; ; zone "10.168.192.in-addr.arpa" type master; file "/home/router/named/rev.10.168.192.in-addr.arpa"; ;

9. Neste momento precisa-se agora configurar o apparmor, permitindo que o sistema fique mais seguro, pois ele proverá, por exemplo, o acesso a porta 53, de DNS, usada pelo Bind9, mas negará a tentativa de abertura de qualquer porta requisitada. Sendo assim, necessita-se editar o arquivo usr.sbin.named contido em /etc/apparmor.d/ adicionando ao final do arquivo, antes de fechar a /usr/sbin/named (contido no arquivo), as seguintes linhas:

```
/home/pi/named/** rw,
/home/pi/named/ rw,
```

Após, deve-se reiniciar o bind9 e o apparmor, utilizando:

```
# /./etc/init.d/apparmor restart
# /./etc/init.d/bind9 restart
```

10. A partir desse momento, têm-se que configurar o serviço de DHCP para que fique automático - pois está manual -, já que o roteador possui os serviços de NAT e DNS devidamente configurados. Como o pacote isc-dhcp-server já foi instalado no começo do capítulo, agora temos que configurar o arquivo /etc/dhcp/dhcpd.conf adicionando o nome de domínio do servidor e alterando o nome do domínio para "lan"conforme configurado anteriormente, configurar também o tempo de *lease* (de concessão) padrão e o máximo de cada IP e é necessário descomentar a linha authoritative:

```
option domain-name "lan";
option domain-name-servers 192.168.0.1 10.0.0.1;
default-lease-time 86400; #Será definido como o período de
um dia.
max-lease-time 172800; #Será definido como um período de
dois dias.
authoritative;
```

11. Agora, no mesmo arquivo do item anterior, precisa-se definir a topologia da rede, como a seguir:

```
subnet 192.168.0.0 netmask 255.255.255.0 {
```

```
range 192.168.0.2 192.168.0.100;
option routers 192.168.0.1;
}
```

12. Tendo configurado, basta reiniciar o isc-dhcp server com o comando:

```
# /./etc/init.d/isc-dhcp-server restart
```

Se der o erro: "dhcpddhcpd *service already running*"basta executar:

```
# rm /run/dhcpd.pid
```

e tentar executar o isc-dhcp-server novamente.

Caso dê algum dos seguintes erros:

- "*No subnet declaration for* wlan0".
- "*No subnet declaration for* eth0".
- "wlan0: *no IPv6 Routers avaiable*".
- "eth0: *no* IPv6 *Routers avaiable*".

é só mudar o arquivo /etc/default/isc-dhcp-server alterando a linha que possui INTERFACESv4= e colocar INTERFACESv4="eth0". Isso resolverá o problema, pois quer dizer que o DHCP só irá “ouvir” a interface eth0.

Anexo II

Instalação e configuração do miniupnpd

Para a instalação do *Deamon* do MiniUPnP, foi necessário baixar o pacote diretamente do github através do comando:

```
$ git clone https://github.com/miniupnp/miniupnp.git
```

E foi necessário também instalar as seguintes dependências:

```
# apt-get install libssl-dev
# apt-get install uuid-dev
```

Depois de tudo baixado e instalado, foi necessário ir para a pasta /miniupnp/miniupnpd/ e executar os seguintes comandos:

```
$ ./genconfig.sh -pcp-peer
```

Para habilitar as operações de MAP e PEER do PCP:

```
$ make -f Makefile.linux
# netfilter/iptables_init.sh
```

Após, altera-se o arquivo miniupnpd.conf nos atributos:

- ext_if_name, alterando para wlan0,
- listening_ip coloca-se eth0,
- ext_ip coloca-se 164.41.75.62, que é o endereço de IP externo,
- enable_natpmp=yes, descomentando essa linha,

- min_lifetime=120, descomentando essa linha,
- max_lifetime=86400, descomentando essa linha.

Depois de realizado esses passos basta agora executar o MiniUPnPd:

```
# ./miniupnpd -f miniupnpd.conf -d
```

Se der o seguinte erro:

```
# ./miniupnpd:  error while loading shared libraries:  li-
bip4tc.so.0:  wrong ELF class:  ELFCLASS64
```

Pode-se resolver instalando o pacote:

```
# apt-get install libevent-2.0-5
```

Já, para a execução do cliente PCP, foi necessário baixar um outro pacote:

```
$ git clone https://github.com/libpcp/pcp.git
```

Nele, navega-se até a pasta /pcp/ e executa os seguintes comandos:

```
$ ./autogen.sh
$ ./configure
$ make
```

Agora, pode-se inicializar o cliente do MiniUPnP para fazer uma operação de MAP do seguinte modo:

```
$ ./pcp -s 192.168.0.1 -i 192.168.0.10:1234 -d
```

Este comando irá fazer uma requisição ao servidor, localizado no IP 192.168.0.1, pedindo-se para abrir a porta 1234 para o cliente que está requisitando, no caso 192.168.0.10.

A resposta do servidor será a seguinte:

```
PCP request received from 192.168.0.10:5350 60bytes.
UPnP permission rule 0 matched :  port mapping accepted
```

```
PCP MAP: Added mapping TCP 1234->192.168.0.10:1234 'PCP MAP
ee3c637cbca7eb62ec017148'
```

Para fazer uma requisição de PEER, deve-se ser feita do seguinte modo:

```
$ ./pcp -s 192.168.0.1 -i 192.168.0.10:1234 -p 8.8.0.0:9 -u
-d
```

A única diferença desse comando para o anterior é o endereço de PEER, que foi dado como 8.8.0.0 na porta 9, definindo o protocolo como UDP (-u) e desabilitando a descoberta de novos servidores PCP (-d).

A resposta do servidor será a seguinte:

```
PCP request received from 192.168.0.10:5350 60bytes
PCP PEER: Added peer mapping UDP 192.168.0.10:1234(1234)-
>8.8.0.0:9 'PCP PEER 74514b7594e299695a464b1f'
```

Realizando com sucesso o processo de MAP e PEER para o roteador. Observando a iptables, temos o seguinte:

```
# iptables -t nat -nL MINIUPNPD
Target = DNAT
prot = tcp
opt = -
source = 0.0.0.0/0
destination = 0.0.0.0/0
tcp dpt:1234 to:192.168.0.10:1234
# iptables -t nat -nL MINIUPNPD-POSTROUTING
Target = SNAT
prot = udp
opt = -
source = 192.168.0.10
destination = 8.8.0.0
udp spt:1234 dpt:9 to:164.41.75.62:1234
```

Se ocorrer o erro UPnP permission rule 4 matched : port mapping rejected, adicionar

no arquivo miniupnp.conf na antepenúltima linha allow 1024-65535 192.168.0.0/24 1024-65535

Anexo III

Instalação do Bacula

Para instalar o *software* Bacula para o servidor, será necessário seguir os seguintes passos:

1. Baixar e descompactar o *software* atualizado do bacula do seguinte site: `https://sourceforge.net/projects/bacula/files/bacula`
2. Instalar o Mysql através do seguinte comando:
   ```
   # apt-get install -y build-essential libreadline6-dev
   zlib1g-dev liblzo2-dev mt-st mtx postfix libacl1-dev
   libssl-dev libmysql++-dev mysql-server
   ```
3. Abra as portas de 9101 à 9103 através do iptables pelo seguinte comando:
   ```
   # iptables -A INPUT -m state -state NEW -m tcp -p tcp -
   dport 9101:9103 -j ACCEPT
   ```
4. Navegue até a pasta extraída do bacula e execute o seguinte comando para a geração e configuração dos arquivos do servidor:
   ```
   # ./configure -with-readline=/usr/include/readline -
   disable-conio -bindir=/usr/bin -sbindir=/usr/sbin -
   with-scriptdir=/etc/bacula/scripts -with-working-
   dir=/var/lib/bacula -with-logdir=/var/log -enable-
   smartalloc -with-mysql -with-archivedir=/mnt/backup
   ```
5. Após, execute o seguinte comando para a compilação, a instalação e a habilitação do início dos *Deamons* em tempo de boot:
   ```
   # make -j8 && make install && make install-autostart
   ```
6. Para a criação do Banco de Dados execute os seguintes comandos:
   ```
   # chmod o+rx /etc/bacula/scripts/
   # /etc/bacula/scripts/create_mysql_database -u root -p
   ```

```
# /etc/bacula/scripts/make_mysql_tables -u root -p
# /etc/bacula/scripts/grant_mysql_privileges -u root -p
```

7. Inicialize os serviços de *Deamon*, de *Storage* e do diretor com o seguinte comando:
   ```
   # service bacula-fd start && service bacula-sd start &&
   service bacula-dir start
   ```

Para instalar o *software* Bacula para o cliente, será necessário seguir os seguintes passos:

1. Baixar e descompactar o *software* atualizado do bacula do seguinte site: `https://sourceforge.net/projects/bacula/files/bacula`
2. Execute o seguinte comando para a instalação de dependências:
   ```
   # apt-get install -y build-essential zlib1g-dev liblzo2-dev
   libacl1-dev libssl-dev
   ```
3. Abra a porta 9102 para a comunicação efetiva do cliente através do iptables através do comando:
   ```
   # iptables -A FW-1-INPUT -m state -state NEW -m tcp -p tcp
   -dport 9102 -j ACCEPT
   ```
4. Navegue até o diretório do Bacula extraído e execute o seguinte comando para a geração e configuração dos arquivos do cliente:
   ```
   # ./configure -enable-client-only -enable-build-dird=no -
   enable-build-stored=no -bindir=/usr/bin -sbindir=/usr/sbin
   -with-scriptdir=/etc/bacula/scripts -with-working-
   dir=/var/lib/bacula -with-logdir=/var/log -enable-
   smartalloc
   ```
5. Após, execute o seguinte comando para a compilação, a instalação e a habilitação do início dos *Deamons* em tempo de *boot*:
   ```
   # make -j8 && make install && make install-autostart
   ```
6. Inicialize o serviço de *Deamon* com o seguinte comando:
   ```
   # service bacula-fd start
   ```

Anexo IV

Configuração do Bacula

O Bacula possui quatro arquivos de configuração essenciais, em que cada um é responsável por determinada função dentro do Bacula. São eles:

- bacula-dir.conf
- bacula-sd.conf
- bacula-fd.conf
- bconsole.conf

O primeiro deles é o principal arquivo de configuração do sistema de *backup*, pois é nele que se são configurados os clientes, os *storages*, as *pools*, os *file sets*, as configurações de retenção e o *schedule*. Já o segundo arquivo, o bacula-sd.conf, será responsável pela definição dos *storages*, configurando assim os armazenamentos de *backup*. O terceiro arquivo, o bacula-fd.conf, será responsável por capturar as informações dos computadores na rede e transmitir para o SD (*Storage Deamon*). E, por fim, o bconsole.conf, que configura o destino para se conectar ao diretor [57].

Assim, para que fosse possível utilizar o *software* Bacula foram necessárias algumas modificações nos arquivos de configuração tanto do servidor quanto do cliente [58] [59].

Sendo assim no arquivo **bacula-dir.conf** do **servidor** foi-se necessário acrescentar ou alterar as seguintes configurações:

Foram modificadas as configurações do diretor, em que foi colocado o nome de VM_Server-dir:

```
Director {
  Name = VM_Server-dir
  DIRport = 9101
  QueryFile = "/etc/bacula/scripts/query.sql"
```

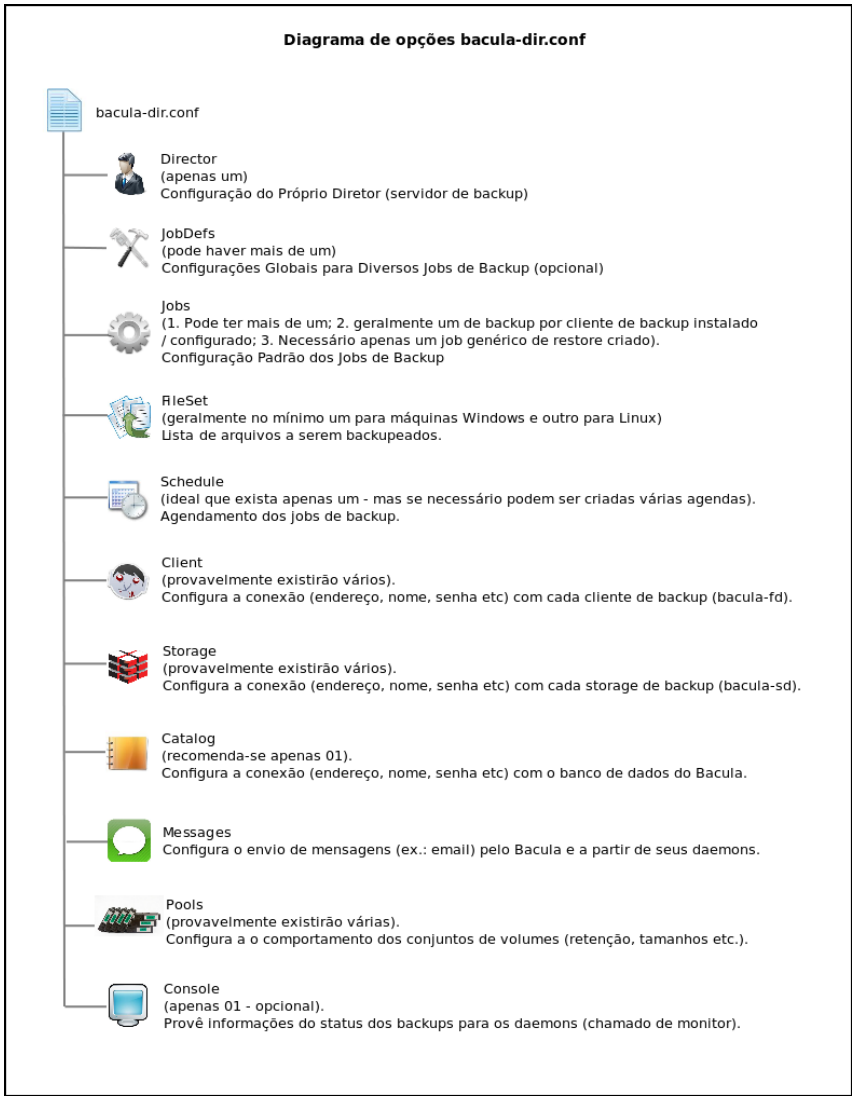

Figura IV.1: Capacidades do arquivo bacula-dir [7].

```
  WorkingDirectory = "/var/lib/bacula"
  PidDirectory = "/var/run"
  Maximum Concurrent Jobs = 20
  Password = "DirectorPWD"
  Messages = Daemon
}
```

Foram adicionadas também duas novas definições de *job* de *backup* para que fossem feitos um *backup* de um único arquivo e um *backup* remoto de arquivos do cliente, sendo respectivamente:

```
JobDefs {
   Name = "ClientFileJob"
   Type = Backup
   Level = Incremental
   Client = VM_Client-fd     #Nome do client
   FileSet = "Backup_BKCPFile"    #Nome do novo Fileset que
será realizado
   Schedule = "WeeklyCycle"
   Storage = File1
   Messages = Standard
   Pool = File
   SpoolAttributes = yes
   Priority = 10
   Write Bootstrap = "/var/lib/bacula/%c.bsr"
}

   JobDefs {
   Name = "ClientFileJobRemote"
   Type = Backup
   Level = Incremental
   Client = VM_Client-fd     #Nome do client
   FileSet = "Backup_BKCPFile_Client"    #Nome do novo Fileset
que será realizado
   Schedule = "WeeklyCycle"
   Storage = File1     #Nome do storage em que será feito o
backup
   Messages = Standard
   Pool = File
   SpoolAttributes = yes
   Priority = 10
   Write Bootstrap = "/var/lib/bacula/%c.bsr"
}
```

Foram adicionados três novos *Jobs* para que fossem realizados o *backup* do arquivo criado, o backup de arquivo do servidor e a restauração dos arquivos remotos, respectivamente:

```
Job {
  Name = "BackupLocalFiles_ForClient"
  Client = VM_Client-fd
  JobDefs = "ClientFileJob"
}

  Job {
  Name = "RemoteBackup_VM_Client"    #Nome do Job criado para
o backup remoto
  JobDefs = "ClientFileJobRemote"    #Definição do Job, que
possui o FileSet de Backup_BCKPFile_Client
  Client = VM_Client-fd     #Cliente a ser requisitado dos ar-
quivos de backup
  Pool = RemoteFile     #Possui uma instância do Remote- }

  Job {
  Name = "RestoreRemote"
  Type = Restore
  Client = VM_Client-fd
  FileSet = "Backup_BCLPFile_Client"
  Storage = File1
  Pool = RemoteFile
  Messages = Standard
  Where = /mnt/backup/bacula-restores }
```

Foram criados também dois novos *FileSet* denominados Backup_BCKPFile e Backup_BCKPFile_Client, adicionando também uma compactação para o arquivo de *backup*:

```
FileSet {
  Name = "Backup_BCKPFile"
  Include {
      Options {
          signature = MD5
          compression = GZIP
      }
```

```
        File = /home/mypcserver/Documents/BCKPFile.txt
    }
    Exclude {
        File = /var/lib/bacula
        File = /mnt/backup
        File = /proc
        File = /tmp
        File = /sys
        File = /.journal
        File = /.fsck
        }
    }

    FileSet {
    Name = "Backup_BCKPFile_Remote"
    Include {
        Options {
            signature = MD5
            compression = GZIP
        }
        File = /home/myhostpc/Documents/
    }
    Exclude {
        File = /var/lib/bacula
        File = /mnt/backup
        File = /proc
        File = /tmp
        File = /sys
        File = /.journal
        File = /.fsck
        }
    }
```

Foi necessária também a criação da configuração de um novo cliente, sendo este o da máquina cliente, para que o servidor reconheça com sucesso o cliente:

```
Client {
```

```
  Name = VM_Client-fd # Nome do client da máquina cliente
  Address = 172.29.52.164 # Endereço IP do cliente
  FDPort = 9102
  Catalog = MyCatalog
  Password = "ClientPWD"# Senha para a comunicação com o cli-
ente
  File Retention = 60 days
  Job Retention = 6 months
  AutoPrune = yes
}
```

Foram necessárias alterações dos campos *Autochanger*, que são as configurações do *storage* do servidor (Os dois arquivos seguem o mesmo estilo, alterando somentes campos *Name*, *Device*, *Media Type* e *Autochanger*):

```
Autochanger {
  Name = File1
  Address = 172.29.3.175
  SDPort = 9103
  Password = "DirectorPWD"
  Device = FileChgr1
  Media Type = File1
  Maximum Concurrent Jobs = 10
  Autochanger = File1 }
```

E, para finalizar as modificações no arquivo do *director*, foi necessária a criação de uma nova *Pool*, somente alterando o campo de formato do *label*, para a posterior facilidade de identificação dos volumes de *backup* remoto.

```
Pool { Name = RemoteFile
  Pool Type = Backup
  Recycle = yes
  AutoPrune = yes
  Volume Retention = 365 days
  Maximum Volume Bytes = 50G
  Maximum Volumes = 100
```

```
   Label Format = "Remote-"}
```

No arquivo **bacula-fd.conf** do **servidor** foi-se necessário alterar as seguintes configurações:

```
Messages {
   Name = Standard
   director = VM_Server-dir = all, !skipped, !restored
}
```

No arquivo **bacula-sd.conf** do **servidor** foi-se necessário alterar as seguintes configurações:

```
Director {
   Name = VM_Server-dir
   Password = "DirectorPWD"
}

   Messages {
   Name = Standard
   director = VM_Server-dir = all
}
```

No arquivo **bconsole.conf** do **servidor** foi-se necessário alterar as seguintes configurações:

```
Director {
   Name = VM_Server-dir
   DIRport = 9101
   address = 172.29.3.175
   Password = "DirectorPWD"
}
```

No arquivo **bacula-fd.conf** do **cliente** foi-se necessário alterar as seguintes configurações:

Mudou-se as configurações do diretor, para que fosse reconhecido o nome do diretor do servidor, assim como a senha, para a autententicação bem sucedida entre o cliente e

o servidor:

```
Director {
   Name = VM_Server-dir
   Password = "ClientPWD"
}
```

Foi necessário também a modificação do *FileDaemon*, gerando as configurações do cliente:

```
FileDeamon {
   Name = VM_Client-fd
   FDPort = 9102
   WorkingDirectory = /var/lib/bacula
   Pid Directory = /var/run
   Maximum Concurrent Jobs = 20
   Plugin Directory = /usr/lib
}
```

Assim como também foi necessário alterar o campo Messages, para que as mensagens de sucesso/erros fossem enviadas para o diretor do servidor:

```
Messages {
   Name = Standard
   director = VM_Server-dir = all, !skipped, !restored
}
```

No arquivo **bconsole.conf** do **cliente** foi necessário alterar as seguintes configurações:

```
Director {
   Name = VM_Server-dir
   DIRport = 9101
   address = 172.29.3.175
   Password = "DirectorPWD"
}
```