UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

LUCAS BERNABÉ GONÇALVES

ESTABELECIMENTO DE CONECTIVIDADE PLENA ENTRE REDES COM IP DE INTRANET: O USO DE VPN WIREGUARD PARA SOLUÇÕES CENTRALIZADAS E DISTRIBUÍDAS

VITÓRIA
2023

LUCAS BERNABÉ GONÇALVES

ESTABELECIMENTO DE CONECTIVIDADE PLENA ENTRE REDES COM IP DE INTRANET: O USO DE VPN WIREGUARD PARA SOLUÇÕES CENTRALIZADAS E DISTRIBUÍDAS

Parte manuscrita do Projeto de Graduação do aluno **Lucas Bernabé Gonçalves**, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Orientador: Dr. Renato Souza Silva
Coorientador: Prof. Dr. Moisés Renato Nunes Ribeiro

VITÓRIA
2023

LUCAS BERNABÉ GONÇALVES

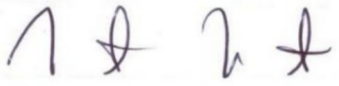
ESTABELECIMENTO DE CONECTIVIDADE PLENA ENTRE REDES COM IP DE INTRANET: O USO DE VPN WIREGUARD PARA SOLUÇÕES CENTRALIZADAS E DISTRIBUÍDAS

Parte manuscrita do Projeto de Graduação do aluno **Lucas Bernabé Gonçalves**, apresentada ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovada em 11 de maio de 2023.

COMISSÃO EXAMINADORA:

Dr. Renato Souza Silva
Orientador



Prof. Dr. Moisés Renato Nunes Ribeiro
Universidade Federal do Espírito Santo
Coorientador

M. Sc. Víctor Manuel García Martínez
Examinador

Edgard da Cunha Pontes
Examinador

VITÓRIA
2023

À minha família, o principal motivo de eu chegar até aqui.

RESUMO

O tunelamento VPN ressurge como base de sistemas SD-WAN por ser uma opção viável técnica e financeiramente para atender às novas demandas de conectividade segura para sistemas descentralizados em nuvem. Nesses sistemas é comum a utilização da topologia de rede em estrela tendo um *hub* centralizador do tráfego, o que pode ser entendido como um limitador, visto que todas as pontas da rede se tornam sensíveis à falhas, sobrecargas e até à localização do mesmo. Para superar esse problema, foi realizada uma pesquisa exploratória com o objetivo de prover uma infraestrutura capaz de armazenar e fornecer os dados necessários para criar conexões peer-to-peer seguras diretamente entre as pontas da comunicação, mesmo quando estas estejam por trás de um NAT. Para criar os túneis, foi utilizado o Wireguard, uma solução VPN vanguardista, de configuração simples, que permite que um dispositivo aja como cliente e servidor ao mesmo tempo. Combinando-o com *wgsd*, um plugin do CoreDNS que utiliza princípios do protocolo STUN, para descoberta do endereço IP e porta públicos correspondentes ao endereço IP e porta privados de um dispositivo em um NAT, e a técnica de *UDP Hole Punching*, para permitir a travessia dos pacotes pelo NAT, foi possível criar conexões *peer-to-peer* entre dois computadores em em redes residenciais distintas. Com o objetivo de testar os ganhos do projeto em termos de redução de latência, foi montada uma estrutura com um servidor atuando tanto como núcleo de roteamento Wireguard quanto como servidor DNS privado (CoreDNS) para a rede Wireguard. No primeiro experimento, este servidor foi instanciado em São Paulo, medindo-se a latência entre os dois clientes passando pelo servidor e diretamente, de acordo com a proposta deste trabalho. Depois deste primeiro experimento, o servidor foi deslocado para Tóquio, repetindo-se as medições de latência. Os resultados encontrados comparando as latências dos pacotes passando pelo servidor e diretamente (*peer-to-peer*) mostram uma redução bastante significativa, da ordem de 25 ms no primeiro experimento, e ainda maior no segundo, com o servidor mais distante, chegando a aproximadamente 600 ms.

Palavras-chave: STUN. *Hole Punching*. WireGuard.

ABSTRACT

VPN tunneling reappears as the basis of SD-WAN systems as it is a technically and financially viable option to meet the new demands for secure connectivity for decentralized cloud systems. In these systems, it is common to use a star network topology with a hub that centralizes traffic, which can be understood as a limiter, since all ends of the network become sensitive to faults, overloads and even its location. To overcome this problem, an exploratory research was carried out with the objective of providing an infrastructure capable of storing and supplying the necessary data to create secure peer-to-peer connections directly between the communication ends, even when they are behind a NAT . To create the tunnels, Wireguard was used, an avant-garde VPN solution, with simple configuration, which allows a device to act as a client and server at the same time. Combining it with wgsd, a CoreDNS plugin that uses principles of the STUN protocol, to discover the public IP address and port corresponding to the private IP address and port of a device in a NAT, and with the technique of UDP Hole Punching, to allow the traversing packets through NAT, it was possible to create peer-to-peer connections between two computers in different home networks. In order to test the project's gains in terms of latency reduction, a structure was set up with a server acting both as a Wireguard routing core and as a private DNS server (CoreDNS) for the Wireguard network. In the first experiment, this server was instantiated in São Paulo, measuring the latency between the two clients passing through the server and directly, according to the proposal of this work. After this first experiment, the server was moved to Tokyo and the latency measurements were repeated. The results comparing the latencies of packets passing through the server and directly (peer-to-peer) show a very significant reduction, on the order of 25 ms in the first experiment, and even greater in the second, with the most distant server, reaching approximately 600 ms.

Keywords: STUN. Hole Punching. WireGuard.

LISTA DE FIGURAS

Figura 1 - Exemplo de topologia proposta utilizando a tecnologia SD-WAN……..7
Figura 2 - Topologia proposta em relação à topologia controle……8
Figura 3 - Interligação de redes privadas de forma segura utilizando a Internet (um meio inseguro).....14
Figura 4 - Exemplo de arquivo de configuração de uma interface Wireguard……... 15
Figura 5 - Operação típica de uma rede com NAT……. 18
Figura 6 - Operação de uma rede com CGNAT……..19
Figura 7 - Processo de *UDP Hole Punching* para dispositivos em NATs diferentes…… 22
Figura 8 - Topologia após *UDP Hole Punching*……22
Figura 9 - Fluxo de comunicações no *UDP Hole Punching*……...23
Figura 10 - Padrão de arquivo de configuração do CoreDNS com o plugin *wgsd*…….25
Figura 11 - Arquivo de configuração da interface de rede *wg0*……..28
Figura 12 - Arquivo de configuração da interface de rede *wg1*……..28
Figura 13 - Arquivo de configuração do CoreDNS…… 29
Figura 14 - Arquivo de configuração de *wg0* em M_1 ……..30
Figura 15 - Arquivo de configuração de *wg1* em M_1……..30
Figura 16 - Arquivo de configuração de *wg0* em M_2……..31
Figura 17 - Arquivo de configuração de *wg1* em M_2 ……. 31
Figura 18 - Estado da interface *wg1* antes da execução do *wgsd-client*…….32
Figura 19 - Estado da interface *wg1* após a execução do *wgsd-client*……32
Figura 20 - Túneis criados utilizando *wg0* e *wg1*……... 33

LISTA DE TABELAS

Tabela 1 - Comparação de latência em milissegundos entre topologias para diferentes localidades do servidor……35

LISTA DE ABREVIATURAS E SIGLAS

AWS	*Amazon Web Services*
CGNAT	*Carrier Grade Network Address Translation*
DNS	*Domain Name System*
DNS-SD	*DNS-Based Service Discovery*
EC2	*Elastic Cloud Compute*
IANA	*Internet Assigned Numbers Authority*
ICE	*Interactive Connectivity Establishment*
IP	*Internet Protocol*
NAT	*Network Address Translation*
P2P	*Peer-to-peer*
RFC	*Request for Comments*
SD-WAN	*Software-Defined Wide Area Network*
SG	*Security Group*
SSH	*Secure Shell*
STUN	*Session Traversal Utilities for NAT*
TCP	*Transmission Control Protocol*
TURN	*Traversal Using Relays around NAT*
UDP	*User Datagram Protocol*
VPN	*Virtual Private Network*

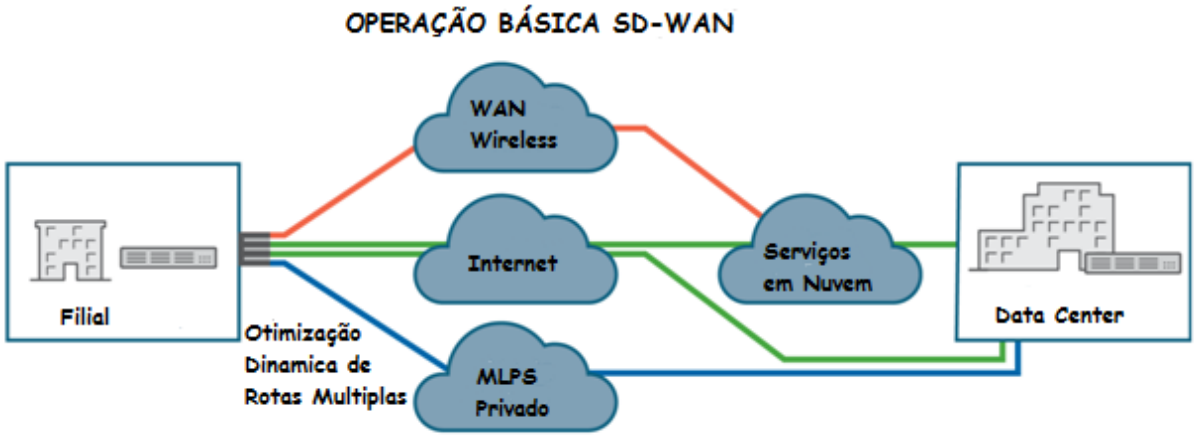
SUMÁRIO

1 Introdução 7
2 Objetivos 11
2.1 Objetivo Geral 11
2.2 Objetivos Específicos 11
3 Referencial teórico 12
3.1 Internet Protocol (IP) 12
3.2 Virtual Private Network (VPN) 13
3.2.1 WireGuard 14
3.3 Software-Defined Wide Area Network (SD-WAN) 16
3.4 Network Address Translation (NAT) 16
3.5 Session Traversal Utilities for NAT (STUN) 20
3.5.1 UDP Hole Punching 21
3.6 DNS-Based Service Discovery (DNS-SD) 23
3.7 wgsd 24
4 Metodologia e etapas de desenvolvimento 26
4.1 Metodologia 26
4.2 Etapas de Desenvolvimento 26
4.2.1 Provisionamento e configuração do servidor 27
4.2.2 Configuração das máquinas finais 29
4.2.3 Alteração da localização do servidor 33
5 Resultados e Análise 35
5.1 Latência 35
5.2 Quantidade de Saltos 35
5.3 Segurança 36
6 Conclusão 37
Referências Bibliográficas 39
APÊNDICE A - GUIA PRÁTICO DE PROVISIONAMENTO DO AMBIENTE DE TESTES 41
APÊNDICE B - RESULTADOS DOS TESTES 46

1 INTRODUÇÃO

A tecnologia de computação em nuvem veio para mudar o paradigma da infraestrutura de armazenamento de dados, computação e conectividade de grandes e pequenas empresas. Grandes *data centers*, que requerem espaço físico, manutenção e oferecem pouca versatilidade, estão sendo trocados por serviços de armazenamento em nuvem feitos sob demanda e altamente elásticos. A abordagem utilizada para prover conectividade independente da tecnologia de transporte de dados, gerenciar tráfego e fazer o controle de acesso nesse caso é conhecida como *Software-Defined Wide Area Network* (SD-WAN), ou seja, uma rede de longa distância definida por software. A Figura 1 ilustra uma arquitetura básica da SD-WAN, interligando uma filial ao *data center* e a outros serviços em nuvem, por soluções de conectividade diferentes.

Figura 1 - Exemplo de topologia proposta utilizando a tecnologia SD-WAN



Fonte: JUNIPER NETWORKS.
Nota: Traduzido pelo autor.

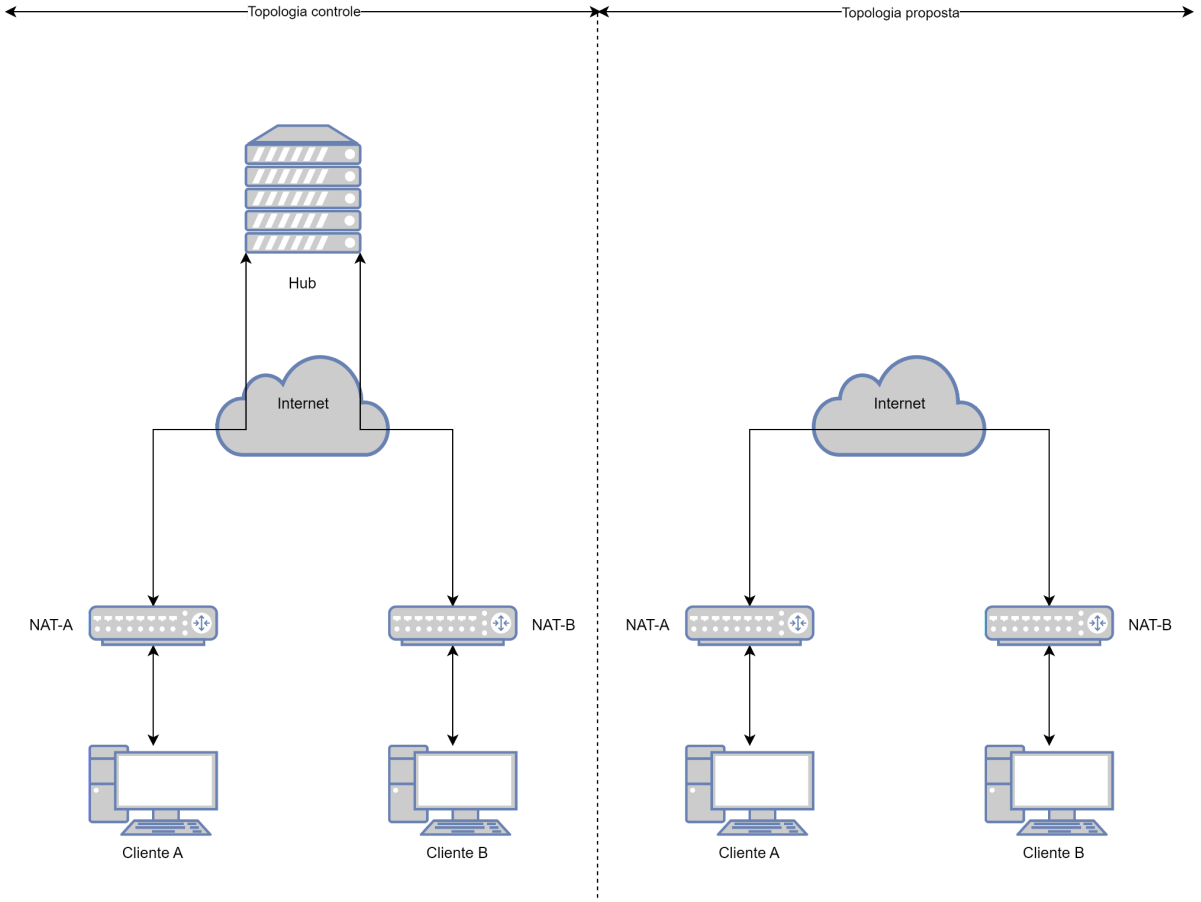
A tecnologia de redes virtuais privadas (VPN, do inglês *Virtual Private Network*) ressurge como forte aliada da SD-WAN, provendo segurança na conectividade, sobre meios inseguros, com a nuvem, onde recursos específicos de roteamento e outras funções de rede estão disponíveis. Mas, ao concentrar os recursos de rede remotamente, acaba-se criando uma topologia em estrela, tendo a nuvem como um *hub* gerenciador do tráfego.

Uma característica desse tipo de topologia é uma grande dependência desse nó central, que está sujeito a falhas de funcionamento, limitação de banda e problemas de conectividade, que

podem ocasionar uma perda de conexão em toda a rede. Outro possível problema é que, dependendo da localização geográfica dos servidores da nuvem e das pontas finais da conexão, pode-se aumentar excessivamente o caminho que os pacotes percorrerão, aumentando assim a latência da conexão. Um exemplo desse problema seria caso os servidores da nuvem estivessem nos Estados Unidos ou na Europa e as pontas finais que necessitam trocar informações estivessem ambas no Brasil.

Nossa proposta para superar esses problemas é utilizar uma aplicação que funcione junto ao SD-WAN e, caso necessário, permita que sejam feitas conexões VPN diretamente entre as pontas, popularmente conhecidas como conexões *Peer-to-peer* (P2P), sem a necessidade de um *hub* centralizador do tráfego.

Figura 2 - Topologia proposta em relação à topologia controle



Fonte: Produção do autor

Para prover uma conectividade segura, utilizaremos o Wireguard, “um túnel de rede seguro, que opera na camada 3” (DONENFELD, 2017, p. 1, Tradução do autor) devido à facilidade

de configuração, uma baixa exigência de processamento, criptografia de ponta e um código relativamente simples, com aproximadamente 4000 linhas de código.

Com foco em auditar a segurança e descobrir possíveis brechas e melhoramentos para o *Wireguard*, Wu (2019) analisou exaustivamente seus protocolos e funcionamento. Uma dessas brechas observadas foi a possibilidade de utilizá-lo em ataques *Denial-of-Service* (DoS) contra terceiros, mas isso não foi relevante o suficiente para não recomendar seu uso em massa.

Os maiores obstáculos são o *Network Address Translation* (NAT) e o *Carrier Grade Network Address Translation* (CGNAT), que são técnicas de tradução de endereços implementada pelas operadoras e provedores de internet devido à escassez de endereços IPv4 enquanto a migração para IPv6 não ocorre por completo. Essa tradução dificulta o alcance de dispositivos externos a um NAT ou CGNAT iniciarem uma sessão com um dispositivo interno à eles. Para superar esse obstáculo, utilizaremos princípios do *Session Traversal Utilities for NAT* (STUN), um protocolo que permite que dispositivos finais (*peers*) em redes com NAT possam descobrir seus IP e porta públicos, sinalizá-los para outro dispositivo, e superar o obstáculo dos NATs utilizando uma técnica chamada *Hole Punching*.

Existem alguns estudos sobre técnicas de travessia de NAT, sobre implementações dessas técnicas para algumas aplicações, mas poucos deles tem foco na aplicação do *Wireguard*. Além disso, existem alguns projetos de implementações do *Wireguard* disponíveis na internet em repositórios como o *GitHub*, mas não há um estudo comparativo sobre a topologia que resulta dessa comunicação *Peer-to-peer* (malha) e uma topologia em estrela.

Em Ford, Srisuresh e Kegel (2005), são estudadas as técnicas de *Hole Punching*, tanto para sessões utilizando *User Datagram Protocol* (UDP) como para conexões utilizando *Transmission Control Protocol* (TCP). O principal foco do estudo é avaliar a eficiência de dois tipos de *Hole Punching* em diferentes fabricantes de dispositivos que realizam o NAT, não sendo considerada a aplicação que se beneficiaria da técnica.

> Enquanto os NATs envolvidos cumpram certos requisitos comportamentais, o *hole punching* funciona consistentemente e robustamente tanto para comunicação TCP quanto para UDP e pode ser implementado por aplicações comuns sem privilégios

> especiais ou informações específicas da topologia de rede. (FORD, SRISURESH, KEGEL, 2005, Tradução do autor)

Em Thu et al. (2014), a técnica de *UDP Hole Punching* foi explorada combinad com o protocolo STUN para viabilizar comunicações P2P atravessando redes com NAT. Além disso, os autores propuseram uma aprimoração no algoritmo do protocolo STUN para testar o tipo de NAT que está envolvido na comunicação.

Em 2013, Dutton publicou um artigo sobre a aplicação do STUN, de alguns protocolos auxiliares como o *Traversal Using Relays around NAT* (TURN) e o *Interactive Connectivity Establishment* (ICE) e do *UDP Hole Punching* para viabilizar a conexão P2P do WebRTC, um cliente de teleconferências online. Nesse artigo são expostos os códigos e plataformas utilizados pelo autor, que podem ser adaptados para a utilização com o Wireguard.

Em seu blog pessoal, Whited (2020) publicou um artigo explorando a possibilidade do Wireguard de criar conexões P2P seguras. Nele, foram explicados os passos para criar uma aplicação baseada no protocolo de Sistema de Nome de Domínio (DNS, do inglês *Domain Name System*) para fornecer os dados de conexão de um *peer* para outro. Com isso, foi possível atualizar a configuração do Wireguard de cada um deles para que o *UDP Hole Punching* seja realizado.

É possível perceber que há uma vasta literatura e provas de conceito sobre a utilização do protocolo STUN e seus princípios, combinado com a técnica de *UDP Hole Punching* para alcançar dispositivos finais por trás NATs. Além disso, temos no Wireguard uma alta confiabilidade em relação à segurança dos dados trafegados e a possibilidade de conexões P2P, devido a sua configuração simples e maleável, em que qualquer dispositivo pode se tornar tanto um servidor como um cliente Wireguard.

Este trabalho propõe utilizar a aplicação desenvolvida por Whited para viabilizar, junto com o Wireguard e a técnica de *UDP Hole Punching*, a conexão P2P segura e escalável entre dispositivos em redes com NAT ou CGNAT. Mesmo que já existam algumas provas de conceito que utilizam esses princípios, a maior contribuição deste trabalho de conclusão de curso é formalizar o conhecimento sobre o assunto e analisar a performance dos túneis criados diretamente em comparação com a topologia que utiliza um centralizador do tráfego.

2 OBJETIVOS

2.1 Objetivo Geral

Ao final deste trabalho esperamos ter desenvolvido uma infraestrutura capaz de sinalizar as requisições de conexão e fornecer os metadados necessários para que uma conexão de VPN *WireGuard* seja estabelecida diretamente entre dois dispositivos finais em um mesmo domínio SD-WAN. O resultado deste trabalho é um sistema completo capaz de criar conexões WireGuard *Peer-to-peer*, além de uma comparação de latência entre a topologia de controle (estrela) e a topologia proposta (malha).

2.2 Objetivos Específicos

- Provisionar um servidor com IP público capaz de receber, armazenar e prover os dados necessários de dispositivos que necessitam criar uma nova conexão Wireguard direta entre os mesmos;
- Criar conexões Wireguard diretas entre dispositivos, mesmo que estejam em redes privadas distintas com NAT aplicado, utilizando o servidor apenas para prover endereços IPs e portas públicos necessários para a configuração do Wireguard nos dispositivos.
- Comparar a latência da comunicação direta proposta com a comunicação utilizando o mesmo servidor como centralizador de tráfego.
- Avaliar as limitações e desvantagens da topologia proposta.

3 REFERENCIAL TEÓRICO

3.1 Internet Protocol (IP)

O Protocolo de Internet, popularmente conhecido como IP (do inglês, *Internet Protocol*), foi desenvolvido para possibilitar a transmissão de dados entre redes de computadores interconectadas. Suas especificações iniciais estão descritas na RFC 791 (POSTEL, 1981), que delimita o escopo do protocolo a "prover as funções necessárias para entregar um pacote de bits (um datagrama de internet) de uma origem para um destino por um sistema interconectado de redes" do que é conhecido como a versão 4 do protocolo.

Nessa versão, que será referida no texto como IPv4, o cabeçalho do datagrama possui de 20 a 60 bytes, sendo que os endereços de origem e destino ocupam 32 bits cada. Sendo assim, os endereços IPv4 podem assumir 2^{32} valores (aproximadamente 4,3 bilhões de endereços possíveis), sendo que aproximadamente 288 milhões de endereços são reservados para propósitos especiais, de acordo com a RFC 6890 (COTTON et al. 2013).

Os endereços IPv4 podem ser divididos entre públicos e privados, sendo os primeiros conhecidos e válidos globalmente na Internet e os segundos apenas conhecidos em redes locais. De acordo com a RFC 1918 (REKHTER et al. 1996), os seguintes intervalos de IP estão reservados pelo Autoridade para Atribuição de Números da Internet (IANA, do inglês *Internet Assigned Numbers Authority*) para redes locais:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

Além disso, a RFC 6598 (WEIL et al. 2012), indica o intervalo 100.64.0.0 - 100.127.255.255 como reservado para provedores de internet utilizarem em redes privadas para compartilhamento de endereços públicos, o que será explicado com maior detalhamento na Seção 4.4.

As funções do protocolo cobrem os propósitos de endereçamento e fragmentação apenas, deixando outras tarefas como controle de fluxo, confiabilidade dos dados, criptografia, entre outros para protocolos complementares.

Considerando que cada dispositivo na internet deve possuir um endereço IP único e que existem projeções de que mais de 27 bilhões de dispositivos estarão conectados à internet até 2025 (PACETE, 2022), fica explícito que o IPv4 não é um protocolo suficiente para garantir o endereçamento de dispositivos à internet. Para solucionar esse problema, foi desenvolvida a versão 6 do Protocolo de Internet (IPv6) que aumenta o tamanho do endereço de 32 para 128 bits (DEERING, HINDEN, 2017), garantindo mais de $3{,}4 \times 10^{38}$ endereços únicos.

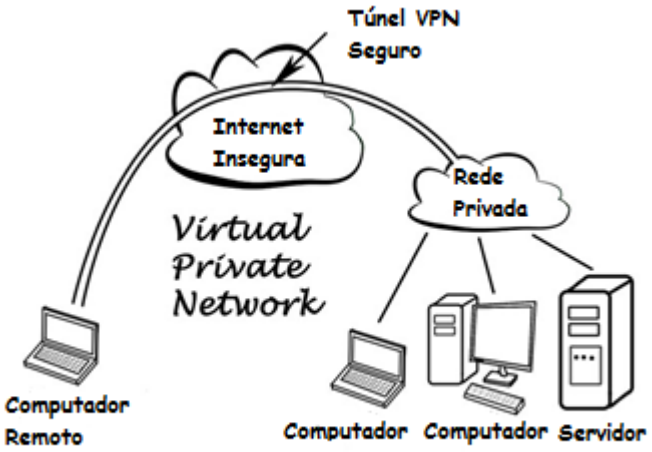
3.2 Virtual Private Network (VPN)

VPN é um acrônimo do inglês para Rede Virtual Privada. Diferente das redes privadas tradicionais, que contam com uma conexão dedicada para a comunicação entre dois ou mais pontos, a VPN utiliza a estrutura da rede pública (Internet) para construir uma comunicação privada e segura entre interlocutores sem restrição de distância geográfica.

> Em VPN, a palavra *Private* corresponde a forma como os dados trafegam, ou seja, os dados são criptografados garantindo a privacidade das informações. O termo *Virtual* indica que as maquinas conectadas na rede não fazem, necessariamente, parte do mesmo meio físico. (BORGES, FAGUNDES, CUNHA, 2014, p. 2)

Apesar de existirem formas de implementação de VPN na camada de enlace e de transporte, nosso trabalho tratará de soluções VPN de camada de rede. Nesse tipo de solução, o pacote IP é criptografado e encapsulado em um novo cabeçalho IP com o ponto onde o pacote será desencapsulado como endereço de destino. Dessa forma é possível garantir a criação de um túnel na rede, aumentando a segurança dos dados e metadados. A Figura 2 mostra uma forma como a VPN pode ser utilizada para interligar redes de forma segura sobre um meio inseguro, a internet.

Figura 3 - Interligação de redes privadas de forma segura utilizando a Internet (um meio inseguro)



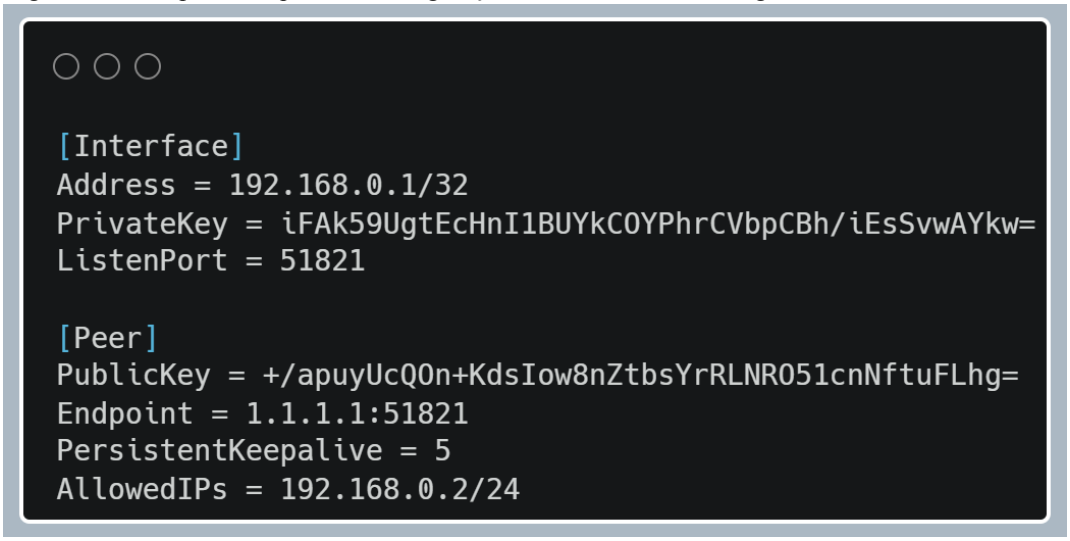
Fonte: MEDIATI (2014).
Nota: Traduzido pelo autor.

3.2.1 WireGuard

WireGuard é um software VPN de código aberto que foi inicialmente apresentado em 2017 por Jason Donanfeld, “mas enquanto o conceito principal continua aplicado, o protocolo evoluiu levemente” (WU, 2019, p. 2, tradução nossa). O protocolo opera na camada de rede, utiliza protocolos de ponta de criptografia e autenticação e utiliza o protocolo UDP na camada de transporte.

De acordo com Donanfeld (2017), o WireGuard foi criado com ambição de substituir outras soluções como IPsec e OpenVPN, sendo mais seguro, mais fácil de usar e entregando maior desempenho. Além disso, ele permite que um mesmo dispositivo seja configurado como servidor e cliente, permitindo assim, conexões P2P. Outro fato importante é que o Wireguard foi integrado ao *kernel* do Linux a partir da versão 5.6 (FISHER, 2020), sendo assim um recurso nativo do sistema operacional.

Figura 4 - Exemplo de arquivo de configuração de uma interface Wireguard.



Fonte: Produção do autor.

Uma forma comum de configurar uma interface de rede do tipo Wireguard é utilizando o comando "*wg-quick up*" com um arquivo de configuração como o ilustrado na Figura 4 no qual a primeira parte (com o identificador "*[Interface]*") expressa a configuração da interface local, enquanto as partes posteriores (identificadas como "*[Peer]*") fazem referência a cada conexão remota nesta interface. Os atributos são os seguintes:

- *Address*: Obrigatório. Endereço ou sub-rede IP privados que a interface assumirá no túnel Wireguard;
- *PrivateKey*: Obrigatório. Chave privada utilizada para realizar, junto à chave pública do *peer* remoto, a criptografia dos pacotes que transitam pelo túnel;
- *ListenPort*: Opcional. Porta UDP privada utilizada para receber e enviar pacotes para o túnel. Caso não seja especificada, a porta 51820 será utilizada por padrão;
- *PublicKey*: Obrigatório. Chave pública utilizada para realizar, junto à chave privada da interface local, a criptografia dos pacotes que transitam pelo túnel. Também serve como identificador do *peer* remoto;
- *Endpoint*: Opcional. Conjunto de endereço IP e porta UDP do *peer* remoto para que a conexão Wireguard seja estabelecida, deve ser conhecido e alcançável pela interface local;
- *PersistentKeepalive*: Opcional. Intervalo de tempo, em segundos, em que pacotes são enviados para manter a sessão ativa caso não haja tráfego de dados pelo túnel;
- *AllowedIPs*: Obrigatório. Endereços ou intervalos de endereços IP de origem, vindos do *peer* remoto, dos quais os pacotes serão aceitos pela interface local. Para aceitar

qualquer pacote do *peer*, basta configurar esse parâmetro como “0.0.0.0/0, ::/0” (IPv4 e IPv6), “0.0.0.0/0” (IPv4) ou “::/0” (IPv6)

3.3 Software-Defined Wide Area Network (SD-WAN)

As SD-WANs são redes de longo alcance definidas por software, esse conceito parte da necessidade das empresas de interconectar suas filiais mesmo estando em localidades geográficas distantes e é estendido pelo paradigma relativamente novo de computação em nuvem. De acordo com Troia et al. (2020), as tecnologias utilizadas em WANs comuns, como o *Multi-Label Protocol Switch* (MPLS), apresentam alguns problemas, tais quais o maior gasto de largura de banda, necessidade de configuração de cada dispositivo separadamente e dificuldade de transição e atualização, dependendo da quantidade de pontos da rede.

Na SD-WAN, toda a gestão, monitoramento e controle da rede é feito por software. Além disso, a infraestrutura de servidores passa a ser um serviço contratado sob demanda, eliminando gastos com manutenção e espaço físico.

3.4 Network Address Translation (NAT)

A sigla NAT (do inglês, *Network Address Translation*) pode ser traduzida como Tradução de Endereços de Rede e compreende uma série de protocolos com o mesmo objetivo: converter endereços IP privados, que só são conhecidos em redes locais, para endereços públicos conhecidos globalmente na internet.

Tendo como base a RFC 2663 (SRISURESH; HOLDREGE, 1999), que descreve as terminologias relacionadas ao NAT, podemos afirmar que todos os tipos de NAT possuem as seguintes características em comum:

- Endereçamento transparente;
- Roteamento transparente pela tradução de endereços;
- Tradução do *payload* do pacote de erro do ICMP.

Devido à falta de endereços IPv4 e a problemas de compatibilidade de dispositivos mais antigos com o IPv6, o NAT é muito utilizado para o compartilhamento de endereços IPv4 públicos durante essa transição. Em um estudo limitado aos usuários da plataforma *Steam*, em

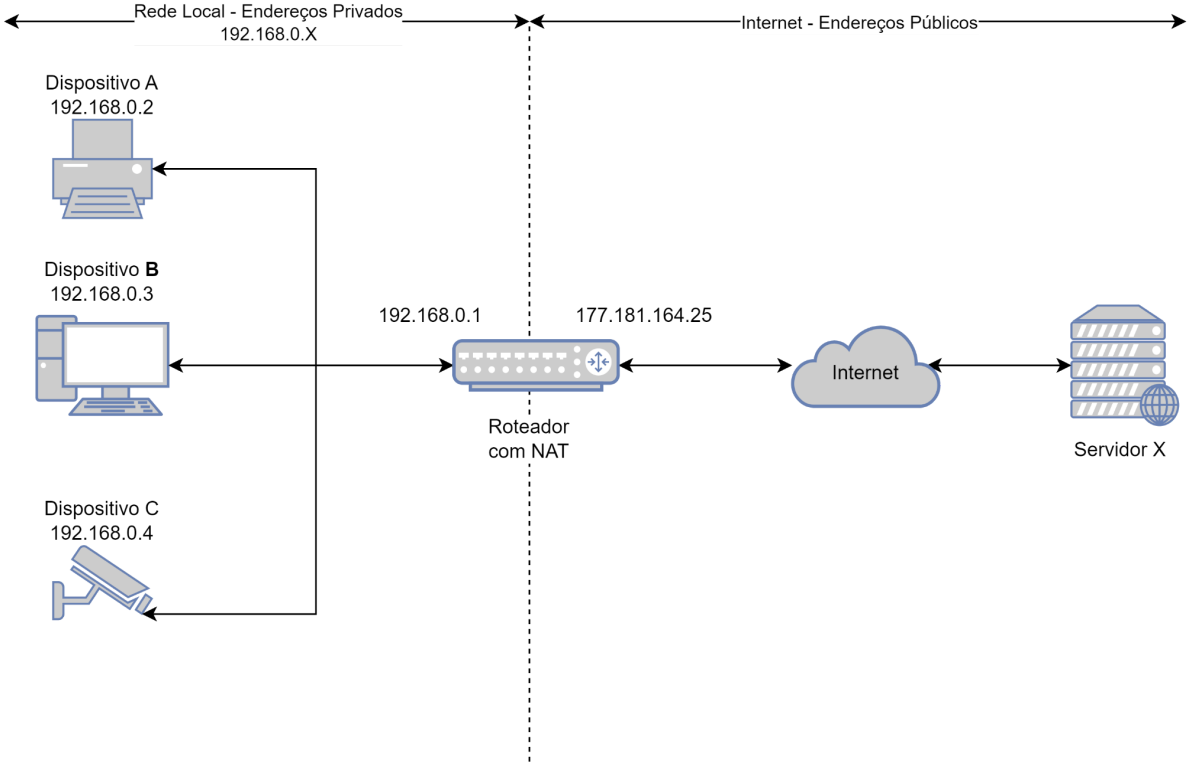
que apenas um usuário pode estar conectado em um dispositivo, estimou-se que em bases de algumas dezenas de milhão de IPs públicos, dezenas de milhares eram compartilhados por pelo menos 10 usuários (ZANDER, MURRAY, 2017).

O tipo de NAT utilizado para realizar esse compartilhamento de IPs públicos é o *Network Address Port Translation* (NAPT), que utiliza dinamicamente identificadores da camada de transporte (chamaremos de portas) concatenadas com o endereço público para multiplexar as conexões entre os múltiplos dispositivos que o compartilham e a Internet.

Tomando como base a Figura 5, para o dispositivo A iniciar uma conexão com um servidor X na internet ele enviará um datagrama com o seu endereço privado (192.168.0.2) como origem no cabeçalho. O roteador NAT receberá esse datagrama e alterará esse cabeçalho, trocando o endereço de origem pelo seu próprio endereço público (177.181.164.25) e associando uma porta a ele (66821, para fins de exemplo). Dessa forma, quando o servidor X enviar uma resposta para essa combinação de endereço e porta (177.181.164.25:66821), o roteador NAT terá informação suficiente para traduzir novamente esse cabeçalho, agora de destino, para 192.168.0.2.

Usando a mesma Figura 5, se o servidor X tentar iniciar uma conexão com qualquer um dos dispositivos na Rede Local, ele enviará um datagrama com o endereço público do roteador NAT como destino, mas sem a informação da porta, que é dinamicamente associada, o roteador NAT não poderá rotear corretamente o datagrama para o destino final. Contudo, caso se tenha controle do roteador NAT, é possível configurá-lo que todos os datagramas que cheguem nele com uma porta de destino predefinida sejam encaminhados para o mesmo destino da Rede Local. Essa técnica é chamada de *Port Forwarding*.

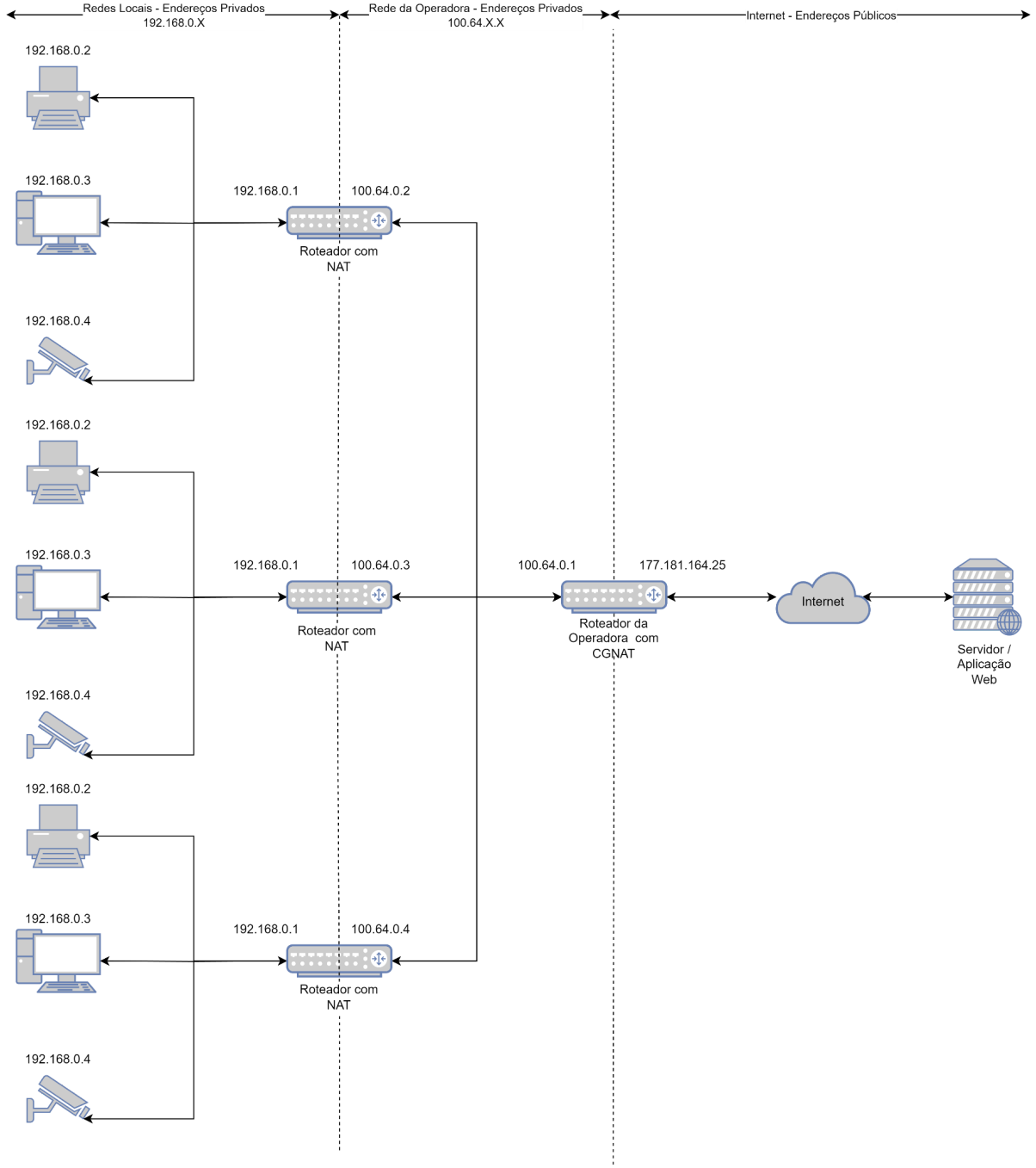
Figura 5 - Operação típica de uma rede com NAT



Fonte: Produção do autor.

Um outro tipo de NAT que vem sendo utilizado pelas operadoras para seguir provisionando internet para os clientes mesmo com falta de endereços IPv4 públicos durante a transição para IPv6 é o *Carrier Grade Network Address Translation* (CGNAT) ou NAT de Larga Escala. Nessa técnica, antes do ponto de acesso ser entregue ao cliente, ele passa por um roteador CGNAT da operadora, que cria uma rede privada da operadora e entrega um IP privado no roteador do cliente. Como o cliente não tem controle do roteador CGNAT, não é possível utilizar o *Port Forwarding* neste caso. A Figura 5 ilustra a topologia de uma rede com CGNAT.

Figura 6 - Operação de uma rede com CGNAT



Fonte: Produção do Autor

O NAT também pode ser classificado de acordo com a implementação do algoritmo de tradução de endereços e portas utilizado. De acordo com Rosenberg et al. (2003), existem quatro variações de NAT.

> Full Cone: um NAT Full Cone é aquele em que todas as requisições de um mesmo par de endereço IP e porta internos são mapeados para uma um mesmo par de endereço IP e porta externos. Além disso, qualquer hospedeiro externo pode enviar

> um pacote para um hospedeiro interno, enviando um pacote para o par externo mapeado.
> Cone Restrito: um NAT de cone restrito é aquele em que todas as requisições de um mesmo par de endereço IP e porta internos são mapeados para uma um mesmo par de endereço IP e porta externos. Diferentemente do NAT Full Cone, um hospedeiro externo (com endereço IP X) pode enviar um pacote para o hospedeiro interno apenas caso o hospedeiro interno tenha enviado previamente um pacote para o endereço X.
> Cone de Porta Restrita: um NAT de cone de porta restrita é como um NAT de cone restrito, mas a restrição inclui os números de porta. Especificamente, um hospedeiro externo pode enviar um pacote com endereço de IP de origem X e porta de origem P para o hospedeiro interno apenas caso o hospedeiro interno tenha enviado previamente um pacote para o endereço IP X e porta P.
> Simétrico: um NAT simétrico é aquele em que todas as requisições de um mesmo par de endereço de IP e porta internos para um par de endereço IP e porta de destino específicos são mapeados para um mesmo par de endereço de IP e porta externos. Se o mesmo hospedeiro enviar um pacote com o mesmo par de endereço e porta de origem, mas para um destino diferente, um mapeamento diferente é utilizado. Assim, apenas o hospedeiro externo que recebeu um pacote pode enviar um pacote UDP de volta para o hospedeiro interno (Rosenberg et al. 2003, Tradução do autor).

3.5 Session Traversal Utilities for NAT (STUN)

Definido em sua versão mais recente na RFC 8489 (PETIT-HUGUENIN et al. 2020), o *Session Traversal Utilities for NAT* (STUN) é um protocolo que "Fornece meios de um dispositivo final determinar o endereço IP e porta alocados por um NAT que correspondem aos seus endereço IP e porta privados" (ROSENBERG, et al., 2008). Para isso, é necessário que exista um servidor STUN com IP público válido e conhecido. Esse servidor é responsável por gerenciar autenticação e armazenar os registros dos clientes, além de fornecer esses dados quando requisitado.

Com a utilização de extensões do STUN, é possível que um dispositivo numa rede com NAT sinalize o conjunto endereço IP e porta públicos para que um dispositivo externo à rede local, inclusive em uma outra rede com NAT, possa tentar iniciar uma conexão com ele. O STUN por si só não pode estabelecer essa conexão, mas ele torna possível uma técnica chamada

Hole Punching. Como o foco deste trabalho é a utilização do Wireguard, que é baseado no protocolo UDP, falaremos do *UDP Hole Punching*.

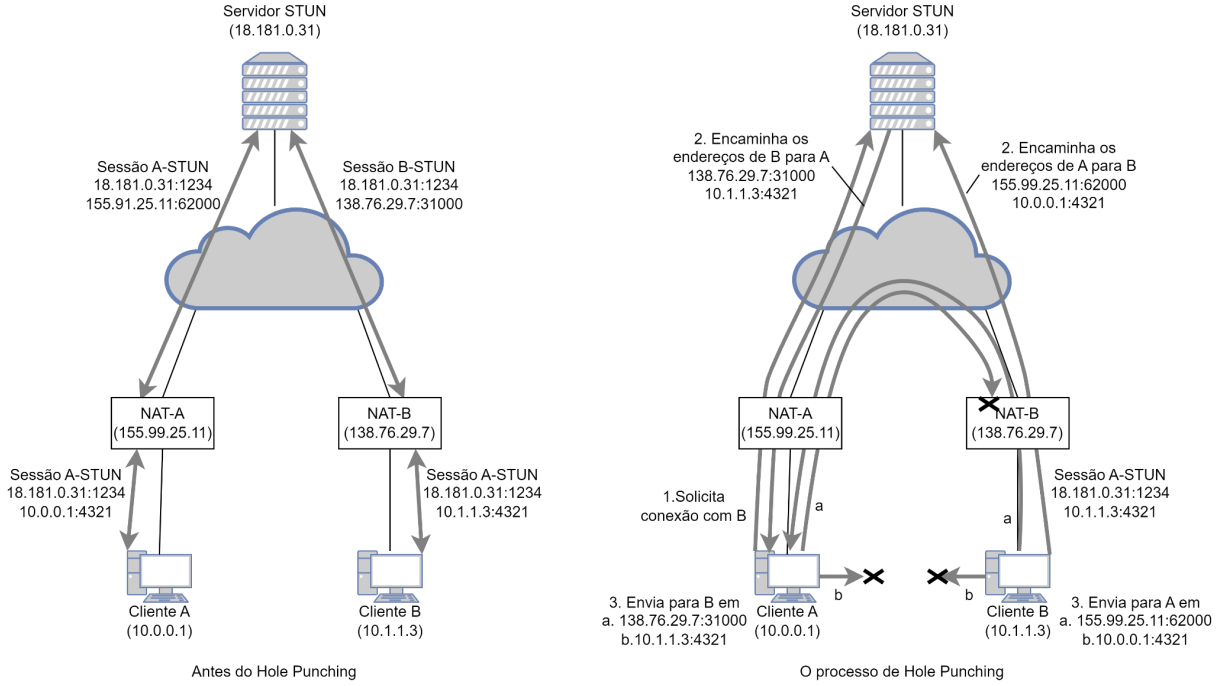
3.5.1 UDP Hole Punching

Para dois dispositivos finais por trás de dois NAT's diferentes se conectarem utilizando a técnica do *UDP Hole Punching*, ambos precisam estar conectados a um terceiro servidor com IP público conhecido, que no caso do exemplo será o servidor STUN. Com essa conexão, o servidor possuirá os IPs e portas públicos e privados dos dois dispositivos.

Adaptando um exemplo de Ford, Srisuresh e Kegel (2005), suponhamos que o dispositivo A deseja estabelecer uma sessão UDP com o dispositivo B e ambos têm sessões ativas com o servidor STUN. O *Hole Punching* se dará da seguinte forma:

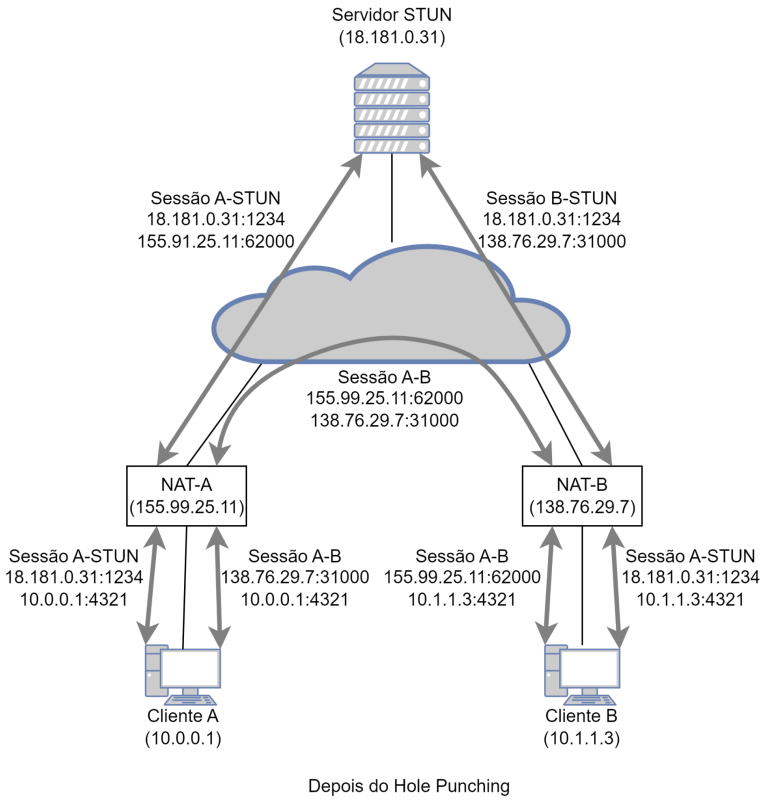
- A inicialmente não sabe como alcançar B, então A pede ajuda ao servidor STUN para estabelecer uma sessão UDP com B;
- O servidor STUN envia uma mensagem para A contendo IP e porta públicos de B junto a um identificador. Ao mesmo tempo o servidor STUN utiliza sua sessão UDP ativa com B para mandar uma mensagem de requisição de conexão para B contendo IP e porta públicos de A;
- Uma vez que essas mensagens são recebidas, A e B conhecem os IPs e portas públicos um do outro;
- A começa a mandar datagramas UDP para o IP e porta pública de B e espera uma resposta válida. Por outro lado, B também começa a enviar datagramas UDP para o IP e porta pública de A. A ordem e tempo dessas mensagens não é crítico.
- Quando A enviar a primeira mensagem para B, o roteador NAT da rede local de A (NAT-A) identificará uma nova sessão sendo criada. A sessão será semelhante a sessão UDP que A tem com o servidor STUN, mas com um destino diferente. Isso faz com que seja aberto um "buraco" no NAT-A para sessões com o IP e porta públicos de B.
- Se a primeira mensagem de B para A já estiver passado pelo roteador NAT da rede local de B (NAT-B) quando a primeira mensagem de A para B chegar ao NAT-B, o mesmo "buraco" estará criado nele, permitindo essa conexão. Caso contrário, a mensagem provavelmente será descartada pelo NAT-B.

Figura 7 - Processo de *UDP Hole Punching* para dispositivos em NAT's diferentes



Fonte: FORD, SRISURESH, KEGEL, 2005.
Nota: Traduzido e adaptado pelo autor.

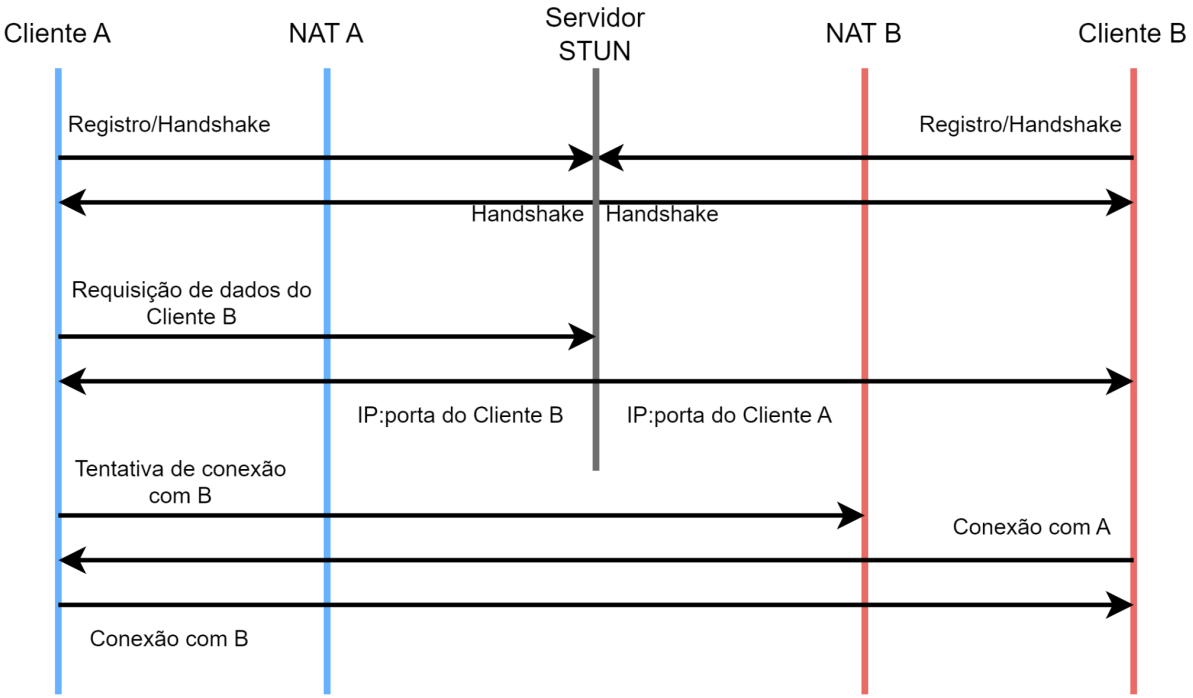
Figura 8 - Topologia após *UDP Hole Punching*



Fonte: FORD, SRISURESH, KEGEL, 2005.
Nota: Traduzido e adaptado pelo autor.

Todo o processo de registro, requisição de conexão e *UDP Hole Punching* utilizando o STUN está resumido na Figura 8.

Figura 9 - Fluxo de comunicações no UDP Hole Punching.



Fonte: Produção do autor.

Essa abordagem possui a limitação de não funcionar com NATs Simétricos, já que o mapeamento identificado pelo servidor STUN só é válido para um destino, que é o próprio servidor. Por exemplo, se o NAT A for simétrico, um mesmo par de endereço IP e porta privados será mapeado diferentemente para o servidor STUN e para o Cliente B e o conjunto de endereço IP e porta públicos que o servidor STUN sinaliza para o Cliente B é inválido para ele.

3.6 DNS-Based Service Discovery (DNS-SD)

A descoberta de serviços baseada em DNS (DNS-SD, do inglês *DNS-Based Service Discovery*) é uma forma de armazenar e identificar serviços em forma de registros de DNS. As especificações de estrutura dos registros são especificadas na RFC 6763, de 2013. “Dado um tipo de serviço que um cliente está procurando, e um domínio no qual o cliente está buscando esse serviço, esse mecanismo permite que clientes descubram uma lista de instâncias do serviço desejado utilizando consultas padrão de DNS” (CHESHIRE; KROCHMAL, 2013, Tradução do autor).

Com objetivo de otimizar as consultas e evitar a duplicidade de registros em caso de mais de uma instância de um mesmo serviço, dois principais tipos de registros são utilizados. Primeiramente, o tipo PTR tem a responsabilidade de armazenar todas as instâncias de um determinado serviço no domínio especificado. Ele possui a estrutura "<serviço>.<domínio>" e quando consultado, retorna todos os registros SRV das instâncias que correspondem à esse serviço. Esse tipo, por sua vez, é estruturado como "<instância>.<serviço>.<domínio>", que quando consultado, retorna a porta e o IP associado à essa instância.

3.7 wgsd

Desenvolvido por Jordan Whited em 2020, o *wgsd* é um plugin para o CoreDNS, uma aplicação de servidor de DNS escrito na linguagem Go. O plugin foi a solução encontrada, utilizando princípios do STUN, para armazenar e servir as informações necessárias para construir uma conexão Wireguard utilizando um protocolo "relativamente simples, maduro (de por volta de 1987), multiplataforma, e acontece de definir um tipo de registro, o registro SRV, para localizar serviços, identificando endereço IP e portas" (WHITED, 2020, Tradução do autor).

> Se estivéssemos escrevendo uma aplicação desde o início que requeresse capacidade de travessia de NAT, o STUN é um componente que deveríamos considerar. Nós não estamos escrevendo o Wireguard, ele já existe, e não é algo que podemos modificar. (lembrando da meta de manter seu código-fonte inalterado) (WHITED, 2020, Tradução do autor).

Quando executado em um servidor Wireguard, especificando-se a interface de rede, o plugin cria registros do tipo SRV para cada *peer* ativo conectado àquela interface. Os registros DNS são criados utilizando as chaves públicas de cada *peer* codificadas em Base32 como identificador de instância, pois, tal como o protocolo de DNS, esse formato não faz diferenciação entre letras maiúsculas e minúsculas enquanto o padrão das chaves, Base64, possui essa diferenciação. Sua configuração segue o seguinte padrão, sendo 5353 a porta em que as consultas são feitas, "lucas.com.br" é o domínio e "*wg1*" é a interface Wireguard de onde as informações dos *peers* serão retiradas:

Figura 10 - Padrão de arquivo de configuração do CoreDNS com o plugin *wgsd*.

Fonte: Produção do autor

Whited também desenvolveu um cliente para realizar as consultas de DNS e, com base nas respostas, atualizar a configuração do Wireguard nas máquinas finais, inserindo o atributo "*Endpoint*" em cada *peer* previamente configurado.

> O wgsd-client é responsável por manter a configuração de endpoint dos peers atualizadas. Ele recupera a lista de peers configurados, consulta o CoreDNS por chaves públicas correspondentes e define os valores de endpoint para cada peer, se necessário (WHITED, 2020, Tradução do autor).

O *wgsd-client*, como é chamado, deve ser executado periodicamente, pois em contratos comuns de provedores de internet é previsto e ocorre a troca do IP público disponibilizado para o contratante de tempos em tempos. Em sua execução, os seguintes parâmetros devem ser passados:

- *device*: o nome da interface Wireguard que deve ser atualizada;
- *dns*: endereço IP e porta do servidor CoreDNS;
- *zone*: domínio configurado no CoreDNS.

4 METODOLOGIA E ETAPAS DE DESENVOLVIMENTO

4.1 Metodologia

A natureza deste trabalho pode ser classificada como uma pesquisa aplicada, já que os resultados poderão ser aplicados em outros trabalhos e em soluções de conectividade para empresas do setor privado e público. Em relação aos objetivos, o tipo de pesquisa empregado no trabalho foi exploratória com uma abordagem quantitativa, pois foi feita a aplicação da teoria existente sobre o assunto para investigar uma possível melhora no desempenho da conexão, utilizando como base as métricas de rede como latência e quantidade de saltos. Os procedimentos técnicos seguiram as características de uma pesquisa experimental, uma vez que foram variadas as topologias de rede para observar o seu efeito sobre as métricas de desempenho das conexões resultantes.

4.2 Etapas de Desenvolvimento

Inicialmente, foi necessário investigar o conteúdo existente sobre a travessia de NAT utilizando o STUN e o *UDP Hole Punching*, buscando as formas de implementação e entendendo suas limitações. Nessa busca, foi encontrado o projeto do *wgsd*, o plugin para CoreDNS exposto na Seção 4.6.

Apesar de não utilizar o protocolo STUN, foi decidido explorar o *wgsd* neste trabalho pela facilidade de configuração e por suas similaridades de princípios aplicados para resolver o mesmo problema. Além disso, o *wgsd* já foi desenvolvido parametrizado com as configurações específicas do Wireguard e conta com o código para ser utilizado nos clientes, atualizando automaticamente as configurações do Wireguard neles, o *wgsd-client*.

Com essa definição, seguiu-se para o provisionamento de um servidor virtual com IP público conhecido, configuração do Wireguard e do CoreDNS com o plugin habilitado e a configuração do Wireguard e do *wgsd-client* em duas máquinas em redes locais com NATs diferentes para a montagem da experiência. Um guia prático de configuração do ambiente de testes está disponível no Apêndice A.

4.2.1 Provisionamento e configuração do servidor

O servidor de DNS foi provisionado utilizando o *Amazon Elastic Compute Cloud* (Amazon EC2), um serviço de computação em nuvem da *Amazon Web Services* (AWS), que permite a criação de máquinas virtuais com IP público gratuitamente por um ano com algumas limitações nas configurações, que não foram impeditivas para o prosseguimento do trabalho. A instância inicial possui uma vCPU e 1 GB de RAM, executa o sistema operacional Ubuntu 20.04 e está localizada na região *sa-east-1*, que corresponde aos datacenters em São Paulo, Brasil.

Cada instância EC2 é necessariamente associada a um Grupo de Segurança (SG, do inglês *Security Group*), que é um conjunto de regras que limitam o acesso à ela para IPs e serviços específicos. Inicialmente as únicas regras de entrada (tráfego para dentro da máquina virtual) desse SG foram a liberação do serviço *Secure Shell* (SSH) para o IP da rede doméstica do autor, para realizar a configuração e a liberação das portas UDP 51820 e 51821 para toda a internet, utilizadas nas interfaces Wireguard. A regra de saída padrão foi mantida, permitindo tráfego da instância para toda a internet.

Para iniciar a configuração, foi instalado o Wireguard e descarregado o arquivo binário do CoreDNS com o plugin *wgsd* no servidor. Para montar o experimento, nele foram configuradas duas interfaces Wireguard: a primeira, que chamaremos de *wg0*, para utilizá-lo na forma mais comum, como um centralizador do tráfego e a segunda (*wg1*) para que o servidor apenas sirva as informações de conexão para as máquinas finais com o *wgsd*. Os arquivos de configuração de *wg0* e *wg1* estão expostos nas Figuras 10 e 11, respectivamente.

Figura 11 - Arquivo de configuração da interface de rede wg0.

```
[Interface]
Address = 10.152.0.254/24
PrivateKey = IAChkdKpNOnL5dJzhW7AFOQupUrwB7H4XerybEUixGQ=
ListenPort = 51821

# Máquina 1
[Peer]
PublicKey = ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
AllowedIPs = 10.152.0.1/32

# Máquina 2
[Peer]
PublicKey = DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
AllowedIPs = 10.152.0.2/32
```

Fonte: Produção do autor.

Figura 12 - Arquivo de configuração da interface de rede wg1.

```
[Interface]
Address = 10.155.0.254/32
PrivateKey = IAChkdKpNOnL5dJzhW7AFOQupUrwB7H4XerybEUixGQ=
ListenPort = 51820

# Máquina 1
[Peer]
PublicKey = ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
AllowedIPs = 10.155.0.1/32

# Máquina 2
[Peer]
PublicKey = DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
AllowedIPs = 10.155.0.2/32
```

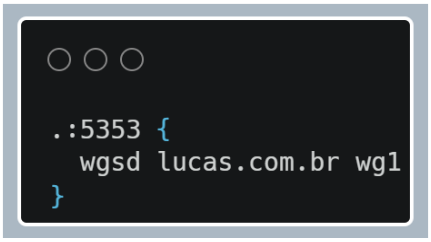
Fonte: Produção do autor.

Pode-se notar duas principais diferenças entre as configurações. A primeira é que a máscara de sub-rede da interface da Figura 10 é /24 enquanto a da Figura 11 é /32, fazendo com que o tráfego possa ser encaminhado por *wg0* para um outro destino, mas não por *wg1*. A segunda diferença são as sub-redes e portas UDP associadas às interfaces, sendo que *wg0* utiliza a porta 51821 e o intervalo 10.152.0.0/24 e *wg1* utiliza a porta 51820 e, apesar de não especificado na configuração (o parâmetro *Address* nesse caso precisa estar com a máscara "/32" para não permitir a retransmissão de pacotes), mas por definição do autor, utiliza a faixa 10.155.0.0/24. Essas configurações, juntamente com as configurações das máquinas finais,

que serão expostas na Seção 5.2.2, permitem comparação em tempo real entre as duas topologias criadas.

Finalizando a configuração das interfaces Wireguard, foi dado o seguimento para a configuração do *wgsd*. O arquivo de configuração do CoreDNS está exposto na Figura 12. É importante salientar que apesar da porta 5353 estar configurada para esse serviço, ela não está exposta por nenhuma regra no SG para assegurar que as consultas de DNS feitas pelos clientes sejam feitas apenas passando pelo túnel Wireguard, mantendo a segurança dos dados de conexão das máquinas finais. Por fim, o servidor foi configurado para executar as interfaces e o serviço CoreDNS na inicialização do sistema operacional.

Figura 13 - Arquivo de configuração do CoreDNS



Fonte: Produção do autor

4.2.2 Configuração das máquinas finais

Ambas as máquinas finais utilizadas no laboratório (chamaremos de M_1 e M_2) utilizam Ubuntu 20.04 como sistema operacional e, apesar de possuírem suporte ao IPv6, tiveram o recurso desabilitado para simular a sua ausência. Em relação à conexão com a internet, M_1 e M_2 estão em redes locais distintas, com IPs públicos distintos, com NATs feitos pelos roteadores das operadoras. Durante os testes e medições, M_1 estava localizada em Vitória e M_2 em Vila Velha, cidades do estado do Espírito Santo, Brasil. O Wireguard foi instalado e o *wgsd-client* foi descarregado nos computadores.

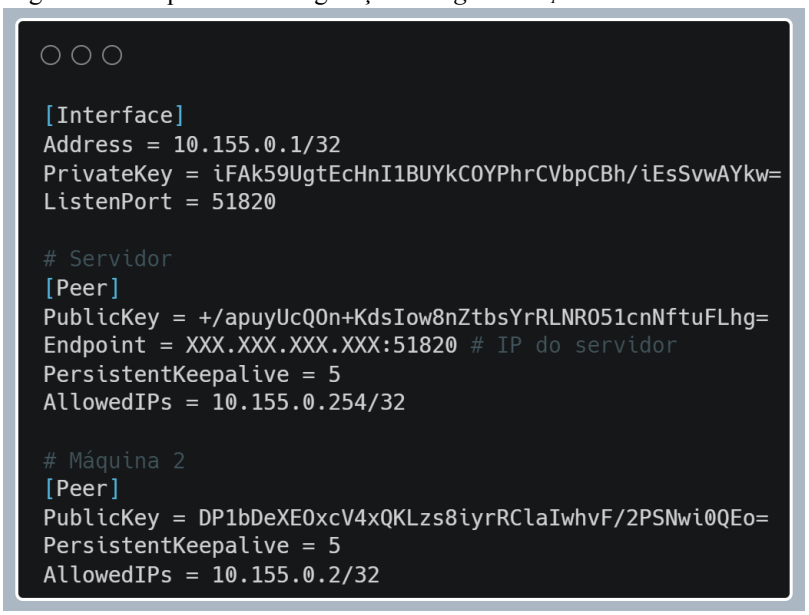
Assim como foi feito no servidor, foram configuradas duas interfaces Wireguard em cada máquina final a fim de criar duas topologias de rede diferentes utilizando os mesmos recursos ao mesmo tempo. As Figuras 13 e 14 expõem, respectivamente, os arquivos de configuração de *wg0* e *wg1* de M_1. As interfaces de M_2 seguem o mesmo padrão, tendo as configurações ilustradas nas Figuras 15 e 16.

Figura 14 - Arquivo de configuração de *wg0* em M_1.

```
[Interface]
Address = 10.152.0.1/24
PrivateKey = iFAk59UgtEcHnI1BUYkCOYPhrCVbpCBh/iEsSvwAYkw=
ListenPort = 51821

# Servidor
[Peer]
PublicKey = +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
Endpoint = XXX.XXX.XXX.XXX:51821 # IP do servidor
PersistentKeepalive = 5
AllowedIPs = 10.152.0.254/24
```

Fonte: Produção do autor.

Figura 15 - Arquivo de configuração de *wg1* em M_1.



```
[Interface]
Address = 10.155.0.1/32
PrivateKey = iFAk59UgtEcHnI1BUYkCOYPhrCVbpCBh/iEsSvwAYkw=
ListenPort = 51820

# Servidor
[Peer]
PublicKey = +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
Endpoint = XXX.XXX.XXX.XXX:51820 # IP do servidor
PersistentKeepalive = 5
AllowedIPs = 10.155.0.254/32

# Máquina 2
[Peer]
PublicKey = DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
PersistentKeepalive = 5
AllowedIPs = 10.155.0.2/32
```

Fonte: Produção do autor.

Figura 16 - Arquivo de configuração de *wg0* em M_2.

```
[Interface]
Address = 10.152.0.2/24
PrivateKey = cnVINXlpUUYzbjA3UWg5bXZRUm04TloxQW9TSUxUbWt=
ListenPort = 51821

# Servidor
[Peer]
PublicKey = +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
Endpoint = XXX.XXX.XXX.XXX:51821 # IP do servidor
PersistentKeepalive = 5
AllowedIPs = 10.152.0.254/24
```

Fonte: Produção do autor.

Figura 17 - Arquivo de configuração de *wg1* em M_2.

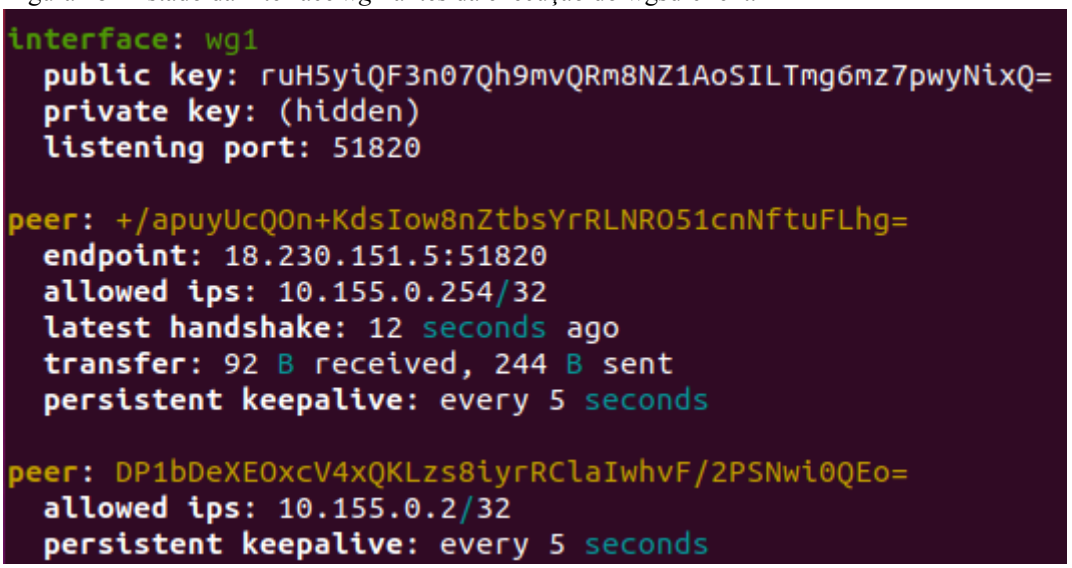
```
[Interface]
Address = 10.155.0.2/32
PrivateKey = cnVINXlpUUYzbjA3UWg5bXZRUm04TloxQW9TSUxUbWt=
ListenPort = 51820

# Servidor
[Peer]
PublicKey = +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
Endpoint = XXX.XXX.XXX.XXX:51820 # IP do servidor
PersistentKeepalive = 5
AllowedIPs = 10.155.0.254/32

# Máquina 1
[Peer]
PublicKey = ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
PersistentKeepalive = 5
AllowedIPs = 10.155.0.1/32
```

Fonte: Produção do autor.

É possível notar que *wg0* é configurado somente para criar as conexões entre cada máquina e o servidor, fazendo com que o tráfego de uma máquina para outra seja necessariamente roteado por ele. Por outro lado, os arquivos de configuração de *wg1*, além de definirem os túneis com o servidor, prevêem uma conexão direta entre as máquinas sem informações de IP e porta, que são requisitadas utilizando o *wgsd-client*.

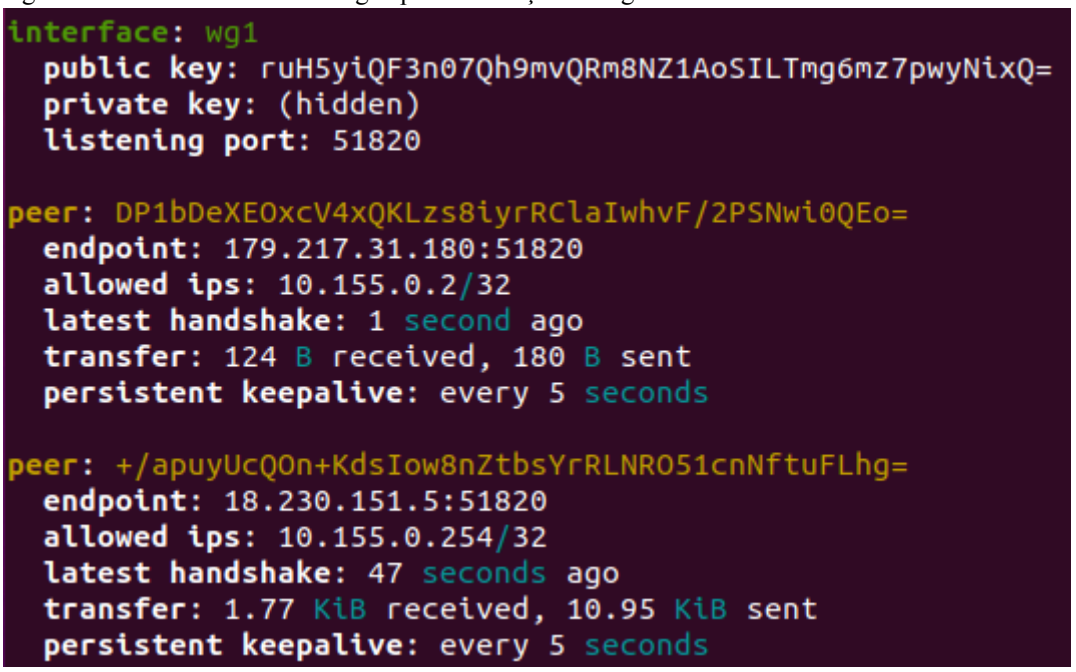
Ao executar o *wgsd-client* em M_1 e M_2, ambas tendo um túnel já estabelecido com o servidor por *wg1*, foi possível estabelecer um túnel Wireguard sem nenhum intermediário entre elas. Por meio das Figuras 18 e 19 é possível visualizar o funcionamento do *wgsd-client* com a visualização do estado da interface *wg1* antes e depois da execução do mesmo.

Figura 18 - Estado da interface *wg1* antes da execução do wgsd-client.



Fonte: Produção do autor.

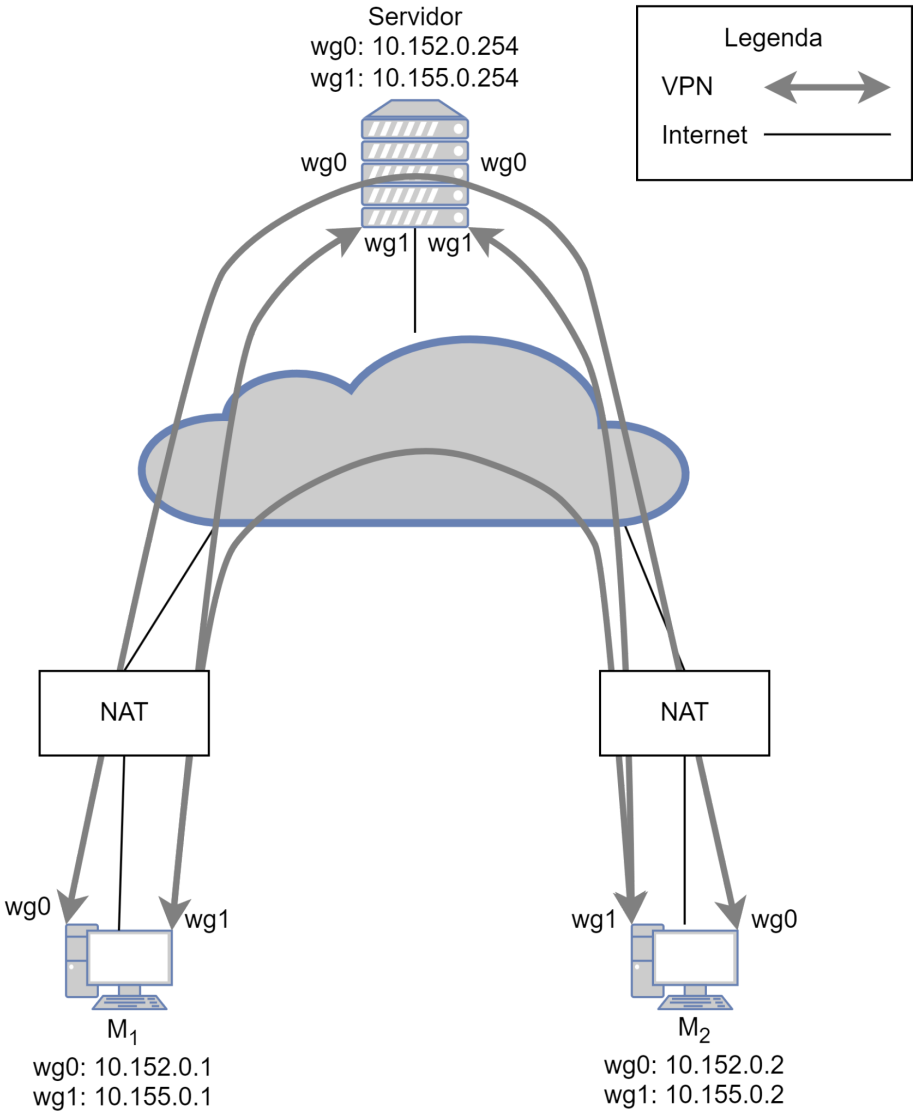
Figura 19 - Estado da interface wg1 após a execução do wgsd-client.



Fonte: Produção do autor.

Sendo assim, temos duas topologias de rede criadas utilizando o servidor, M_1 e M_2. Com as interfaces *wg0* de cada dispositivo, foi estabelecida uma topologia em estrela, tendo o servidor como nó central. Já as interfaces *wg1* dos dispositivos foram utilizadas para criar uma malha, com conexão direta entre as pontas da comunicação. A Figura 20 ilustra os túneis criados para cada interface.

Figura 20 - Túneis criados utilizando *wg0* e *wg1*.



Fonte: Produção do autor.

O comando "*fping*" foi utilizado para a medição da latência (o tempo, em milissegundos, que um pacote leva para ir à um destino e voltar à sua origem) e o comando "*traceroute*" foi usado para contagem de saltos feitos pelos pacotes na comunicação entre as máquinas finais em ambas as topologias e reafirmar suas características. Para a latência, a medição foi feita com dois disparos de 20 pacotes no mesmo intervalo de tempo para *wg0* e *wg1*. O primeiro partindo de M_1 com destino a M_2 e o segundo fazendo o caminho inverso.

4.2.3 Alteração da localização do servidor

Para evidenciar a dependência da localização do servidor na topologia em estrela e contrastar com a topologia proposta, foi criada uma cópia do servidor em uma região diferente de

datacenters da AWS, sendo ela a *ap-northeast-1* (Tóquio, Japão). Os mesmos procedimentos expostos na Seção 5.2.2 foram reproduzidos nesses servidores e os mesmos testes de medição de latência e quantidade de saltos foram realizados.

5 Resultados e Análise

5.1 Latência

A Tabela 1 apresenta os resultados do teste de latência para *wg0* e *wg1* com a utilização dos servidores de cada localidade. É possível afirmar que as conexões de M_1 e M_2 com a internet estão estáveis durante os testes, pois não há uma variação significativa das latências máximas e mínimas em relação à média.

Tabela 1 - Comparação de latência em milissegundos entre topologias para diferentes localidades do servidor

	wg0			***wg1***		
	mínima	média	máxima	mínima	média	máxima
São Paulo	48,5	57,05	65,8	13,4	25,05	35,2
Japão	609	620	635	11,8	25,9	33,6

Fonte: Produção do autor

Pode-se afirmar que a topologia proposta em malha apresentou uma redução de 56,09% (32 ms) de latência média em relação à topologia mais comumente utilizada, em estrela, com o servidor localizado em São Paulo. Ao alterar a localização do servidor para o Japão, esse ganho de performance foi para 95,82% (594,1 ms). É importante ressaltar que essa alteração não gerou diferenças significativas nas conexões em malha, já que nesse caso o servidor é utilizado apenas para auxiliar a criação dos túneis entre as máquinas finais.

5.2 Quantidade de Saltos

Ao utilizar o comando *traceroute* com o IP privado dentro do túnel Wireguard, o comportamento foi o mesmo para ambas as localidades dos servidores, para a topologia associada a *wg0*, a comunicação se dá em dois saltos enquanto para *wg1*, apenas um salto é dado pelo pacote. Apesar dessa uniformidade, o teste passando por dentro do túnel não reflete a realidade do caminho feito pelo pacote. Na verdade, ao sair de *wg0* de M_1, o pacote chega ao servidor e é encaminhado à *wg0* de M_2 de forma encapsulada, o que explica a alta latência de cada salto.

Esses resultados têm uma relação direta com o exposto na Seção 5.1, pois, por mais que o caminho do pacote dentro de qualquer um dos túneis com o servidores seja representado da

mesma forma, a realidade é que esses túneis são estabelecidos passando por ativos de rede comuns dos provedores de internet. Dessa forma, fica explícito que a localização dos recursos envolvidos na comunicação tem um alto impacto na mesma, por isso é importante prover os meios para haja possibilidade de criar uma conexão direta entre os *peers*, como feito neste trabalho.

5.3 Segurança

Além do ganho em performance, é importante enfatizar que o nível de segurança da topologia proposta se mantém em relação à topologia de controle. No servidor, por conta do Grupo de Segurança, apenas as portas UDP 51820 e 51821 foram abertas para conexão com a Internet. Essas portas estão relacionadas ao serviço Wireguard, que carrega consigo toda segurança implementada em seu desenvolvimento.

Dessa forma, para que um computador remoto tenha acesso ao servidor, é necessário que sua chave pública esteja configurada em uma das interfaces Wireguard dele. O mesmo ocorre com as máquinas finais, que só estabelecem túneis conforme a configuração feita previamente.

Em um cenário produtivo, supondo que o servidor esteja em um datacenter próprio de uma empresa, a responsabilidade de restringir conexões às portas do mesmo deve ser de um dispositivo de Firewall, substituindo as regras do SG da AWS.

6 CONCLUSÃO

Este trabalho foi proposto para utilizar o Wireguard, uma aplicação VPN, para criar conexões *Peer-to-Peer* seguras entre dispositivos de um mesmo domínio SD-WAN transpassando os NATs aos quais os *peers* podem estar sujeitos, gerando uma topologia de rede em malha e assim fazendo um contraponto com a topologia em estrela, mais tradicional no SD-WAN. O objetivo foi criar uma rede mais resiliente e eliminar o nó centralizador de tráfego, resultando também em uma menor latência entre os peers da comunicação.

Para isso, foi utilizado *wgsd*, um projeto de Whited (2020) que funciona como plugin do CoreDNS, uma aplicação de servidor de DNS, para armazenar IPs públicos e portas de Wireguard dos *peers*. Esses, por sua vez, utilizaram o *wgsd-client*, uma aplicação complementar também desenvolvida por Whited, para consultar as informações de outros peers e, com base na resposta dessas consultas, atualizar sua própria configuração do Wireguard para que o processo de *UDP Hole Punching* se inicie. Dessa forma, foi possível estabelecer os túneis diretamente entre dois computadores em redes diferentes, com NAT aplicado. Ao mesmo tempo, a topologia em estrela foi mantida, tendo a máquina virtual que funcionou como servidor de DNS também agindo como nó central do SD-WAN. As topologias contrastantes foram criadas utilizando os mesmos recursos computacionais, diferenciando-se apenas pelas interfaces de rede e intervalos de endereço IP privado. Enquanto a topologia em estrela ficou associada à interface *wg0* e à rede privada 10.152.0.0/24, a malha foi alocada na interface *wg1* e na rede 10.155.0.0/24.

Com toda a infraestrutura de rede privada virtual construída, foram feitas as medições de latência com o comando "*fping*" e de quantidade de saltos com o comando "*traceroute*" para comunicações entre os *peers*. Com isso, pôde-se afirmar que a topologia proposta teve um ganho de performance, traduzido em redução de latência média, em 56,09%. Isso pode ser explicado pela quantidade de saltos dados pelos pacotes, pois enquanto um pacote enviado por *wg0* deu 2 saltos, um pacote enviado por *wg1* levou 1 apenas salto para chegar ao mesmo destino, tendo a mesma origem.

Para ressaltar o contraste de dependência da localização do servidor nas duas topologias, o mesmo foi migrado de São Paulo para o Japão. A diferença foi aumentada entre as topologias.

Como esperado, a latência média do túnel direto entre os *peers* se manteve no mesmo patamar, mas o ganho de performance em relação ao túnel que tem o servidor como centralizador de tráfego foi de 95,82%.

Pode-se afirmar que a topologia em malha criada com o Wireguard, auxiliada pelo *wgsd*, é promissora em relação à latência, que só depende da localização geográfica e da conexão com a Internet das duas partes da comunicação. Uma outra vantagem percebida é a redução do impacto de uma possível falha ou indisponibilidade do servidor, visto que após os túneis já estarem estabelecidos, ele só precisará ser utilizado caso o IP público de algum *peer* seja alterado. Além disso, é possível configurar o Wireguard de forma com que as pontas dos túneis diretos sejam centralizadores de tráfego de outras redes privadas, podendo assim conectar diversos equipamentos por túnel de forma segura.

Um ponto negativo dessa abordagem é o aumento da complexidade da rede, pois dessa forma a gerência de roteamento e das regras de segurança é feita de forma distribuída. Isso torna necessário planejar e configurar toda a infraestrutura de forma minuciosa para não gerar *loops*, assimetrias de rede ou qualquer outro problema relacionado à regras de roteamento inconsistentes entre si. Além disso, essa característica faz com que em uma rede com muitos nós, a adição de mais um nó cause a necessidade de alterar as configurações do Wireguard de todos os nós já existentes. Como projeto futuro, é sugerido o desenvolvimento de um serviço de gerenciamento centralizado de rede, capaz de criar e fornecer novos arquivos de configuração para todos os nós da rede a partir da adição ou remoção de um nó.

Como citado na Seção 3.5.1, uma limitação da abordagem tratada é que alguns tipos de NAT, principalmente o NAT Simétrico, não são compatíveis com o *UDP Hole Punching*. Existem alguns estudos que tratam esse problema com técnicas de previsão de tradução de porta. A implementação dessas técnicas como complemento ao *wgsd* é uma sugestão de trabalho futuro.

Referências Bibliográficas

BORGES, Fábio; FAGUNDES, Bruno Alves; DA CUNHA, Gerson Nunes. Vpn: Protocolos e segurança. **S/D**, v. 10, 2019.

CHESHIRE, Stuart; KROCHMAL, Marc. RFC 6763: DNS-based service discovery. **Internet Engineering Task Force**, p. 1-49, 2013.

COTTON, M. et al. RFC 6890. **Special-purpose IP address registries**. 2013.

DEERING, Steve; HINDEN, Robert. RFC 8200. **Internet protocol, version 6 (IPv6) specification**. 2017.

DONENFELD, Jason A. WireGuard: Next Generation Kernel Network Tunnel. In: Network and Distributed System Security Symposium, 2017, San Diego. **Proceedings**[…]. San Diego: NDSS, 2017.

DUTTON, Sam. **WebRTC in the real world:** STUN, TURN and signaling. **Google**, Nov, p. 1-22, 2013.

FISHER, Dennis. **WireGuard VPN Added to Linux Kernel**. In: **DUO**. 30 mar. 2020. Disponível em: https://duo.com/decipher/wireguard-vpn-added-to-linux-kernel. Acesso em: 21 maio 2023.

FORD, Bryan; SRISURESH, Pyda; KEGEL, Dan. Peer-to-Peer Communication Across Network Address Translators. In: **USENIX Annual Technical Conference, General Track**. 2005. p. 179-192.

JUNIPER NETWORKS. **What is SD-WAN.** Disponível em: https://www.juniper.net/uk/en/products-services/what-is/sd-wan/. Acesso em: 22 nov. 2020.

MEDIATI, Nick. Everything You Need to Know About VPNs. *In*: **TECHSOUP**. 12 dez. 2014. Disponível em: https://www.techsoup.org/support/articles-and-how-tos/everything-you-need-to-know-about-vpns. Acesso em: 22 nov. 2020.

PACETE, Luiz Gustavo. IoT: até 2025, mais de 27 bilhões de dispositivos estarão conectados. **Forbes**, 11 ago. 2022. Disponível em: https://forbes.com.br/forbes-tech/2022/08/iot-ate-2025-mais-de-27-bilhoes-de-dispositivos-estarao-conectados/. Acesso em: 21 set. 2022.

PETIT-HUGUENIN, Marc et al. RFC 8489. **Session Traversal Utilities for NAT (STUN)**. 2020.

POSTEL, Jon. RFC 791. **Internet protocol**. 1981.

REKHTER, Yakov et al. RFC 1918. **Address allocation for private internets**. 1996.

ROSENBERG, Jonathan et al. RFC 3489. **STUN-simple traversal of user datagram protocol (UDP) through network address translators (NATs)**. 2003.

SRISURESH, Pyda; HOLDREGE, Matt. RFC 2663. **IP network address translator (NAT) terminology and considerations**. 1999.

THU, Ha Tran Thi et al. Combining stun protocol and udp hole punching technique for peer-to-peer communication across network address translation. *In:* **2014 International Conference on IT Convergence and Security (ICITCS)**. IEEE, 2014. p. 1-4.

TROIA, Sebastian, MOREIRA ZORELLO, Ligia M.; MARALIT, Alvin J.; MAIER, Guido. SD-WAN: An Open-Source Implementation for Enterprise Networking Services. *In*: INTERNATIONA CONFERENCE ON TRANSPARENT OPTICAL NETWORKS (ICTON), 22., 2020, Bari. **Proceedings** […]. Bari: IEEE, 2020. p. 1-4.

WEIL, Jason et al. RFC 6598. **IANA-reserved IPv4 prefix for shared address space**. 2012.

WHITED, Jordan. WireGuard Endpoint Discovery and NAT Traversal using DNS-SD. *In:* Jordan Whited. **Jordan Whited Personal Blog.** 20 mai. 2020. Disponível em: https://www.jordanwhited.com/posts/wireguard-endpoint-discovery-nat-traversal/. Acesso em: 4 dez. 2022.

WU, Peter. **Analysis of the WireGuard protocol**. 2019. Tese (Mestrado)**.** Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, 2019.

ZANDER, Sebastian; MURRAY, David. **Share or Not**: Investigating the Presence of Large-Scale Address Sharing in the Internet. In: **2017 IEEE 42nd Conference on Local Computer Networks (LCN)**. IEEE, 2017. p. 243-251.

APÊNDICE A - GUIA PRÁTICO DE PROVISIONAMENTO DO AMBIENTE DE TESTES

- 1ª Etapa - Provisionamento da máquina virtual com IP público:
    - Após criar uma conta na AWS e fazer login no console de administração, selecione a região mais próxima (no caso do Brasil, selecionar a região sa-east-1);
    - Busque pelo serviço EC2 (Elastic Cloud Computing);
    - Clique em “Instâncias” no menu lateral esquerdo;
    - Clique no botão “Executar Instâncias”;
    - Crie uma instância com as seguintes especificações:
        - Sistema Operacional: Ubuntu Server 20.04 LTS (HVM), SSD Volume Type;
        - Arquitetura: 64 bits (x86);
        - Tipo da instância: t2.micro;
        - Par de chaves: Crie um novo par de chaves (RSA/.pem), faça download da chave e selecione-a;
        - Grupo de segurança: Permitir tráfego SSH de “Meu IP”;
        - Mantenha os outros atributos como padrão.
    - Após executar a instância, deve-se voltar ao painel de instâncias, selecioná-la e, na aba “Segurança”, clicar no Grupo de Segurança associado à ela.
    - Na aba, “Regras de entrada”, clique no botão “Editar regras de entrada”;
    - Após clicar no botão “Adicionar regra”, selecione o tipo “UDP Personalizado”, adicione “51820-51821” no campo “Intervalo de portas” e “0.0.0.0/0” no campo “Origem”;
    - Clique no botão “Salvar regras”.

- 2ª Etapa - Configuração inicial do servidor e dos peers:
    - Em um computador com sistema operacional Linux, abra o terminal e faça login no servidor virtual utilizando o comando: “ssh -i /caminho/da/chave.pem ubuntu@IP_púlico_do_servidor” (Não é necessário fazer nos peers);
    - Utilize o comando “sudo apt-get install wireguard” para fazer download do Wireguard;
    - Utilize o comando “sudo apt-get install curl” para fazer download do CURL;

- Utilize o comando “curl -K -o https://github.com/jwhited/wgsd/releases/download/v0.3.3/wgsd_0.3.3_linux_amd64.tar.gz” para fazer download dos binários do CoreDNS e do wgsd-client em um arquivo compactado;
- Utilize o comando “tar -xvf wgsd_0.3.3_linux_amd64.tar.gz” para descompactar o arquivo;
- Utilize o comando “wg genkey > private” para gerar uma chave privada e depois “cat private” para ver o seu conteúdo. Guarde o resultado;
- Utilize o comando “wg pubkey < private” para visualizar a chave pública resultante. Guarde o resultado.

- 3ª Etapa - Configuração do Wireguard no Servidor.
  - Utilize o comando “sudo su” para elevar permissões de administrador;
  - Utilize o comando “nano /etc/sysctl.conf” para editar o arquivo sysctl.conf, descomentando a linha “net.ipv4.ip_forward=1”(utilize Ctrl+x para salvar). Execute o comando “sysctl -p” para aplicar as alterações. Isso permitirá o roteamento de pacotes pelo Servidor;
  - Utilize o comando “nano /etc/wireguard/wg0.conf” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
[Interface]
Address = 10.152.0.254/24
PrivateKey = Chave privada do servidor
ListenPort = 51821

# Máquina 1
[Peer]
PublicKey = Chave pública da Máquina 1
AllowedIPs = 10.152.0.1/32

# Máquina 2
[Peer]
PublicKey = Chave pública da Máquina 2
AllowedIPs = 10.152.0.2/32
```

  - Utilize o comando “nano /etc/wireguard/wg1.conf” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
[Interface]
Address = 10.155.0.254/32
PrivateKey = Chave privada do Servidor
ListenPort = 51820

# Máquina 1
[Peer]
PublicKey = Chave pública da Máquina 1
AllowedIPs = 10.155.0.1/32

# Máquina 2
[Peer]
PublicKey = Chave pública da Máquina 2
AllowedIPs = 10.155.0.2/32
```

- Utilize os comandos “systemctl enable wg-quick@wg0.service” e “systemctl daemon-reload” para que a interface wg0 seja habilitada para iniciar junto à inicialização do sistema;
- Utilize o comando “systemctl start wg-quick@wg0” para iniciar a interface wg0 imediatamente;
- Repita os dois passos anteriores para wg1.

- 4ª Etapa - Configuração do CoreDNS no Servidor.
  - Utilize o comando “cp coredns /usr/local/bin/coredns” para copiar o binário do CoreDNS para o diretório em que ele será executado;
  - Utilize o comando “nano /usr/local/bin/start-coredns.sh” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
#!/bin/bash

./usr/local/bin/coredns -conf=/usr/local/bin/Corefile
```

  - Utilize o comando “nano /usr/local/bin/Corefile” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
.:5353 {
  wgsd dominio.com.br wg1
}
```

  - Utilize o comando “nano /etc/systemd/system/start-coredns.service” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
[Unit]
After=wg-quick@wg1.service

[Service]
ExecStart=/usr/local/bin/start-coredns.sh

[Install]
WantedBy=default.target
```

- Utilize o comando “chmod 744 /usr/local/bin/start-coredns.sh” para alterar as permissões do arquivo “start-coredns.sh”;
- Utilize o comando “chmod 664 /etc/systemd/system/start-coredns.service” para alterar as permissões do arquivo “start-coredns.service”;
- Utilize os comandos “systemctl daemon-reload” e “systemctl enable start-coredns.service” para iniciar o CoreDNS junto à inicialização do sistema;
- Reinicie o Servidor.

- 5ª Etapa - Configuração dos peers.
  - Utilize o comando “sudo su” para elevar permissões de administrador;
  - Utilize o comando “nano /etc/sysctl.conf” para editar o arquivo sysctl.conf, adicionando as linhas listadas abaixo. Execute o comando “sysctl -p” para aplicar as alterações. Isso desabilitará o IPv6 da máquina;
    - net.ipv6.conf.all.disable_ipv6 = 1
    - net.ipv6.conf.default.disable_ipv6 = 1
    - net.ipv6.conf.lo.disable_ipv6 = 1
  - Utilize o comando “nano /etc/wireguard/wg0.conf” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
[Interface]
Address = 10.152.0.1/24 #Caso seja Máquina 2, alterar para 10.152.0.2/24
PrivateKey = Chave privada da Máquina
ListenPort = 51821

# Servidor
[Peer]
PublicKey = Chave pública do Servidor
Endpoint = XXX.XXX.XXX.XXX:51821 # XXX.XXX.XXX.XXX=IP do servidor
PersistentKeepalive = 5
AllowedIPs = 10.152.0.254/24
```

  - Utilize o comando “wg-quick up wg0” para iniciar a interface wg0;

- Utilize o comando “nano /etc/wireguard/wg1.conf” para criar um arquivo com o seguinte conteúdo (utilize Ctrl+x para salvar):

```
[Interface]
Address = 10.155.0.1/32 #Caso seja Máquina 2 trocar para 10.155.0.2/32
PrivateKey = Chave privada da Máquina
ListenPort = 51820

# Servidor
[Peer]
PublicKey = Chave pública do Servidor
Endpoint = XXX.XXX.XXX.XXX:51820 # XXX.XXX.XXX.XXX=IP do servidor
PersistentKeepalive = 5
AllowedIPs = 10.155.0.254/32

# Máquina 2 (ou Máquina 1)
[Peer]
PublicKey = Chave pública da outra Máquina
PersistentKeepalive = 5
AllowedIPs = 10.155.0.2/32 #Caso seja Máquina 2 trocar para 10.155.0.1/32
```

- Utilize o comando “wg-quick up wg1” para iniciar a interface wg1;
- Utilize o comando “./wgsd-client -device=wg1 -dns=10.155.0.254:5353 -zone=dominio.com.br” para atualizar as informações de conexão do peer oposto.
- Observação: os arquivos de configuração de wg0 e wg1 estão no padrão da Máquina 1, deve-se atentar às observações feitas nos comentários (após o símbolo “#”).

APÊNDICE B - RESULTADOS DOS TESTES

Aqui estão expostos os resultados dos testes realizados no dia 22 de dezembro de 2022 remotamente, utilizando as redes residenciais dos participantes. Participaram desses testes o autor, Lucas Bernabé Gonçalves, e Renato Souza Silva, co-orientador deste trabalho. Os resultados foram copiados da saída do terminal dos computadores, sem tratamento, sendo M_1 máquina utilizada por Lucas e M_2 por Renato. Por já haver uma configuração prévia do Wireguard na máquina do Renato, o nome *wg0* foi substituído por renato *e wg1* foi substituído por *renato_mesh*.

M1:

```
root@C1496:/home/lucasgoncalves# nano /etc/wireguard/wg1.conf
root@C1496:/home/lucasgoncalves# nano /etc/wireguard/wg0.conf
root@C1496:/home/lucasgoncalves# sysctl -p
net.ipv6.conf.all.disable_ipv6 = 1
net.ipv6.conf.default.disable_ipv6 = 1
net.ipv6.conf.lo.disable_ipv6 = 1
root@C1496:/home/lucasgoncalves# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
Warning: AllowedIP has nonzero host part: 10.152.0.254/24
[#] ip -4 address add 10.152.0.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
root@C1496:/home/lucasgoncalves# wg-quick up wg1
[#] ip link add wg1 type wireguard
[#] wg setconf wg1 /dev/fd/63
[#] ip -4 address add 10.155.0.1/32 dev wg1
[#] ip link set mtu 1420 up dev wg1
[#] ip -4 route add 10.155.0.254/32 dev wg1
[#] ip -4 route add 10.155.0.2/32 dev wg1
root@C1496:/home/lucasgoncalves# wg
interface: wg0
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 12 seconds ago
  transfer: 92 B received, 392 B sent
  persistent keepalive: every 5 seconds

interface: wg1
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
```

```
  listening port: 51820

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 12 seconds ago
  transfer: 92 B received, 244 B sent
  persistent keepalive: every 5 seconds

peer: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  allowed ips: 10.155.0.2/32
  persistent keepalive: every 5 seconds
root@C1496:/home/lucasgoncalves# ./Downloads/wgsd-client -device=wg1
-dns=10.155.0.254:5353 -zone=lucas.com.br
2022/12/22 08:19:25 [+/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=] no SRV records
found
root@C1496:/home/lucasgoncalves# wg
interface: wg0
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 48 seconds ago
  transfer: 1.08 KiB received, 10.79 KiB sent
  persistent keepalive: every 5 seconds

interface: wg1
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51820

peer: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  endpoint: 179.217.31.180:51820
  allowed ips: 10.155.0.2/32
  latest handshake: 1 second ago
  transfer: 124 B received, 180 B sent
  persistent keepalive: every 5 seconds

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 47 seconds ago
  transfer: 1.77 KiB received, 10.95 KiB sent
  persistent keepalive: every 5 seconds
root@C1496:/home/lucasgoncalves# fping -a -A -c 20 10.152.0.2 10.155.0.2

10.152.0.2 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 49.4/57.9/64.6
```

```
10.155.0.2 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 13.4/25.6/35.2
root@C1496:/home/lucasgoncalves#
root@C1496:/home/lucasgoncalves# fping -a -A -c 20 10.152.0.2 10.155.0.2
10.152.0.2 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 49.7/56.5/61.7
10.155.0.2 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 18.5/25.3/32.9
root@C1496:/home/lucasgoncalves# traceroute 10.155.0.2
traceroute to 10.155.0.2 (10.155.0.2), 30 hops max, 60 byte packets
 1  10.155.0.2 (10.155.0.2)  21.671 ms  21.643 ms  31.381 ms
root@C1496:/home/lucasgoncalves# traceroute 10.152.0.2
traceroute to 10.152.0.2 (10.152.0.2), 30 hops max, 60 byte packets
 1  10.152.0.254 (10.152.0.254)  22.281 ms  22.267 ms  22.254 ms
 2  10.152.0.2 (10.152.0.2)  57.954 ms  57.940 ms  57.923 ms
root@C1496:/home/lucasgoncalves# wg-quick down wg0
[#] ip link delete dev wg0
root@C1496:/home/lucasgoncalves# wg-quick down wg1
[#] ip link delete dev wg1
root@C1496:/home/lucasgoncalves# nano /etc/wireguard/wg0.conf
root@C1496:/home/lucasgoncalves# nano /etc/wireguard/wg1.conf
root@C1496:/home/lucasgoncalves# wg-quick up wg0
[#] ip link add wg0 type wireguard
[#] wg setconf wg0 /dev/fd/63
Warning: AllowedIP has nonzero host part: 10.152.0.254/24
[#] ip -4 address add 10.152.0.1/24 dev wg0
[#] ip link set mtu 1420 up dev wg0
root@C1496:/home/lucasgoncalves# wg-quick up wg1
[#] ip link add wg1 type wireguard
[#] wg setconf wg1 /dev/fd/63
[#] ip -4 address add 10.155.0.1/32 dev wg1
[#] ip link set mtu 1420 up dev wg1
[#] ip -4 route add 10.155.0.254/32 dev wg1
[#] ip -4 route add 10.155.0.2/32 dev wg1
root@C1496:/home/lucasgoncalves# wg
interface: wg0
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 8 seconds ago
  transfer: 92 B received, 212 B sent
  persistent keepalive: every 5 seconds

interface: wg1
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51820

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
```

```
  endpoint: 3.112.55.39:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 5 seconds ago
  transfer: 92 B received, 212 B sent
  persistent keepalive: every 5 seconds

peer: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  allowed ips: 10.155.0.2/32
  persistent keepalive: every 5 seconds
root@C1496:/home/lucasgoncalves# ./Downloads/wgsd-client -device=wg1
-dns=10.155.0.254:5353 -zone=lucas.com.br
2022/12/22 08:28:05 [+/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=] no SRV records
found
root@C1496:/home/lucasgoncalves# wg
interface: wg0
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 1 minute, 3 seconds ago
  transfer: 92 B received, 564 B sent
  persistent keepalive: every 5 seconds

interface: wg1
  public key: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  private key: (hidden)
  listening port: 51820

peer: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  endpoint: 179.217.31.180:51820
  allowed ips: 10.155.0.2/32
  latest handshake: 20 seconds ago
  transfer: 188 B received, 212 B sent
  persistent keepalive: every 5 seconds

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 1 minute ago
  transfer: 796 B received, 884 B sent
  persistent keepalive: every 5 seconds
root@C1496:/home/lucasgoncalves# fping -a -A -c 20 10.152.0.2 10.155.0.2
10.152.0.2 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 609/619/635
10.155.0.2 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 18.1/26.5/33.6
```

M_2:

```
root@rssilva-UX430UAR:/etc/wireguard# vim renato.conf
root@rssilva-UX430UAR:/etc/wireguard# vim renato_mesh.conf
root@rssilva-UX430UAR:/etc/wireguard# wg-quick up renato.conf
wg-quick: `/etc/wireguard/renato.conf.conf' does not exist
root@rssilva-UX430UAR:/etc/wireguard# wg-quick up renato
[#] ip link add renato type wireguard
[#] wg setconf renato /dev/fd/63
Warning: AllowedIP has nonzero host part: 10.152.0.254/24
[#] ip -4 address add 10.152.0.2/24 dev renato
[#] ip link set mtu 1420 up dev renato
root@rssilva-UX430UAR:/etc/wireguard# wg-quick up renato_mesh
[#] ip link add renato_mesh type wireguard
[#] wg setconf renato_mesh /dev/fd/63
[#] ip -4 address add 10.155.0.2/32 dev renato_mesh
[#] ip link set mtu 1420 up dev renato_mesh
[#] ip -4 route add 10.155.0.254/32 dev renato_mesh
[#] ip -4 route add 10.155.0.1/32 dev renato_mesh
root@rssilva-UX430UAR:/etc/wireguard# wg
interface: wg0
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 56504

peer: NtXFN0HBJ+KgpjmXnq8SZgMaAx94KpanbpP5n7Wajlc=
  endpoint: 177.221.123.123:51820
  allowed ips: 10.168.0.0/16, 10.50.0.0/16
  latest handshake: 12 minutes, 30 seconds ago
  transfer: 154.67 MiB received, 32.15 MiB sent

interface: renato
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 40 seconds ago
  transfer: 92 B received, 404 B sent
  persistent keepalive: every 5 seconds

interface: renato_mesh
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51820

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 18 seconds ago
```

```
  transfer: 92 B received, 276 B sent
  persistent keepalive: every 5 seconds

peer: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  allowed ips: 10.155.0.1/32
  persistent keepalive: every 5 seconds

root@rssilva-UX430UAR:/etc/wireguard#                    /home/rssilva/Downloads/wgsd-client
-device=renato_mesh -dns=10.155.0.254:5353 -zone=lucas.com.br
2022/12/22  08:19:21  [+/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=]  no  SRV  records
found
root@rssilva-UX430UAR:/etc/wireguard# wg
interface: wg0
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 56504

peer: NtXFN0HBJ+KgpjmXnq8SZgMaAx94KpanbpP5n7Wajlc=
  endpoint: 177.221.123.123:51820
  allowed ips: 10.168.0.0/16, 10.50.0.0/16
  latest handshake: 18 minutes, 23 seconds ago
  transfer: 154.67 MiB received, 32.15 MiB sent

interface: renato
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 24 seconds ago
  transfer: 368 B received, 3.08 KiB sent
  persistent keepalive: every 5 seconds

interface: renato_mesh
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51820

peer: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  endpoint: 179.234.228.251:1043
  allowed ips: 10.155.0.1/32
  latest handshake: 1 second ago
  transfer: 180 B received, 420 B sent
  persistent keepalive: every 5 seconds

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 18.230.151.5:51820
  allowed ips: 10.155.0.254/32
```

```
  latest handshake: 4 seconds ago
  transfer: 1.05 KiB received, 3.27 KiB sent
  persistent keepalive: every 5 seconds
root@rssilva-UX430UAR:/etc/wireguard# fping -a -A -c 20 10.152.0.1 10.155.0.1
10.152.0.1 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 48.5/56.2/65.8
10.155.0.1 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 14.3/24.5/33.2
root@rssilva-UX430UAR:/etc/wireguard# vim renato.conf
root@rssilva-UX430UAR:/etc/wireguard# vim renato_mesh.conf
root@rssilva-UX430UAR:/etc/wireguard# wg-quick down renato_mesh
[#] ip link delete dev renato_mesh
root@rssilva-UX430UAR:/etc/wireguard# wg-quick down renato
[#] ip link delete dev renato
root@rssilva-UX430UAR:/etc/wireguard# wg-quick up renato_mesh
[#] ip link add renato_mesh type wireguard
[#] wg setconf renato_mesh /dev/fd/63
[#] ip -4 address add 10.155.0.2/32 dev renato_mesh
[#] ip link set mtu 1420 up dev renato_mesh
[#] ip -4 route add 10.155.0.254/32 dev renato_mesh
[#] ip -4 route add 10.155.0.1/32 dev renato_mesh
root@rssilva-UX430UAR:/etc/wireguard# wg-quick up renato
[#] ip link add renato type wireguard
[#] wg setconf renato /dev/fd/63
Warning: AllowedIP has nonzero host part: 10.152.0.254/24
[#] ip -4 address add 10.152.0.2/24 dev renato
[#] ip link set mtu 1420 up dev renato
root@rssilva-UX430UAR:/etc/wireguard# wg
interface: wg0
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 56504

peer: NtXFN0HBJ+KgpjmXnq8SZgMaAx94KpanbpP5n7Wajlc=
  endpoint: 177.221.123.123:51820
  allowed ips: 10.168.0.0/16, 10.50.0.0/16
  latest handshake: 26 minutes, 37 seconds ago
  transfer: 154.67 MiB received, 32.15 MiB sent

interface: renato_mesh
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51820

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 9 seconds ago
  transfer: 92 B received, 212 B sent
  persistent keepalive: every 5 seconds

peer: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
```

```
  allowed ips: 10.155.0.1/32
  persistent keepalive: every 5 seconds

interface: renato
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 3 seconds ago
  transfer: 92 B received, 180 B sent
  persistent keepalive: every 5 seconds
root@rssilva-UX430UAR:/etc/wireguard#                    /home/rssilva/Downloads/wgsd-client
-device=renato_mesh -dns=10.155.0.254:5353 -zone=lucas.com.br
2022/12/22  08:28:25  [+/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=]  no  SRV  records
found
root@rssilva-UX430UAR:/etc/wireguard# wg
interface: wg0
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 56504

peer: NtXFN0HBJ+KgpjmXnq8SZgMaAx94KpanbpP5n7Wajlc=
  endpoint: 177.221.123.123:51820
  allowed ips: 10.168.0.0/16, 10.50.0.0/16
  latest handshake: 27 minutes, 25 seconds ago
  transfer: 154.67 MiB received, 32.15 MiB sent

interface: renato_mesh
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51820

peer: ruH5yiQF3n07Qh9mvQRm8NZ1AoSILTmg6mz7pwyNixQ=
  endpoint: 179.234.228.251:1043
  allowed ips: 10.155.0.1/32
  latest handshake: 22 seconds ago
  transfer: 212 B received, 220 B sent
  persistent keepalive: every 5 seconds

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51820
  allowed ips: 10.155.0.254/32
  latest handshake: 57 seconds ago
  transfer: 796 B received, 852 B sent
  persistent keepalive: every 5 seconds

interface: renato
```

```
  public key: DP1bDeXEOxcV4xQKLzs8iyrRClaIwhvF/2PSNwi0QEo=
  private key: (hidden)
  listening port: 51821

peer: +/apuyUcQOn+KdsIow8nZtbsYrRLNRO51cnNftuFLhg=
  endpoint: 3.112.55.39:51821
  allowed ips: 10.152.0.0/24
  latest handshake: 51 seconds ago
  transfer: 92 B received, 468 B sent
  persistent keepalive: every 5 seconds
root@rssilva-UX430UAR:/etc/wireguard# fping -a -A -c 20 10.152.0.1 10.155.0.1
10.152.0.1 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 615/621/626
10.155.0.1 : xmt/rcv/%loss = 20/20/0%, min/avg/max = 11.8/25.3/32.3
```

Houve um outro teste no dia 14 de dezembro de 2022 no Espaço Empreendedor da Ufes com a presença de Renato Souza Silva (co-orientador) e do Prof. Dr. Moisés Renato Nunes Ribeiro com um dos computadores conectado à rede cabeada da Ufes enquanto o outro utilizava a conexão 4G roteada de um celular. Os resultados desse teste foram desconsiderados devido à instabilidade do sinal da rede 4G utilizada.