Universidade de São Paulo
Instituto de Matemática e Estatística
Bacharelado em Ciência da Computação

Desenvolvimento de um sistema descentralizado de envio de mensagens

Uma análise das dificuldades e limitações de sistemas descentralizados

João Renner Rudge

Monografia Final

mac 0499 — Trabalho de Formatura Supervisionado

Supervisor: Prof. Dr. Daniel Macêdo Batista

São Paulo
2024

Agradecimentos

Let's see what happens.
— Unnamed fictional character from Randall Munroe's *What if?*

À minha família, por sempre me apoiar nos meus estudos, me educar, me permitir chegar onde estou hoje; ao computador do escritório da minha mãe, por cativar minha curiosidade e me ensinar uma infinitude de coisas ao longo da minha infância; aos meus amigos, por sempre estarem do meu lado, fazendo companhia, me dando algo para fazer todo dia.

Resumo

João Renner Rudge. **Desenvolvimento de um sistema descentralizado de envio de mensagens: *Uma análise das dificuldades e limitações de sistemas descentralizados***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2024.

Sistemas distribuídos são parte essencial da Internet, possibilitando a troca de mensagens, arquivos, ou até mesmo dinheiro entre pessoas ao redor do mundo. Tais sistemas podem ser desenhados de forma centralizada ou descentralizada, a depender da existência de uma entidade central que coordene as interações entre os participantes. Enquanto sistemas descentralizados não possuem ponto central de falha, eles possuem uma série de dificuldades e desafios técnicos para sua implementação. Este trabalho realiza uma análise de protocolos descentralizados de envio de mensagens modernos, classificando-os de acordo com suas arquiteturas. Em seguida, é proposta a implementação de um protótipo funcional que combina as melhores características desses protocolos para permitir a troca de mensagens entre usuários na Internet. O protótipo desenvolvido é compatível com configurações de rede comuns, permite que os usuários utilizem nomes de usuário simples, e alterna entre diferentes formas de conexão dependendo das condições de rede que os usuários se encontram. Além disso, técnicas de criptografia e de roteamento por caminhos aleatórios são afixados para garantir a privacidade e confidencialidade das mensagens trocadas.

Palavras-chave: Sistemas descentralizados. Protocolos de mensagens. Criptografia. Resiliência a falhas.

Abstract

João Renner Rudge. **Development of a descentralized messaging system: *An analysis of the difficulties and limitations of descentralized systems***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2024.

Distributed systems are an essential part of the Internet, enabling the exchange of messages, files, and even money between individuals worldwide. These systems can be designed in either a centralized or decentralized manner, depending on the presence of a central entity that coordinates interactions among participants. While decentralized systems lack a single point of failure, they present numerous technical difficulties and implementation challenges. This work conducts an analysis of modern decentralized messaging protocols, classifying them based on their architectures. Subsequently, the implementation of a functional prototype is proposed, combining the best features of these protocols to facilitate message exchange between Internet users. The developed prototype is compatible with common network configurations, allows users to employ simple usernames, and dynamically switches between different connection methods depending on network conditions. Additionally, encryption techniques and random path routing are employed to ensure the privacy and confidentiality of exchanged messages.

Keywords: Descentralized systems. Messaging protocols. Cryptography. Fault tolerance.

Lista de abreviaturas

DNS	Sistema de Nomes de Domínio (*Domain Name System*)
CDN	Rede de Distribuição de Conteúdo (*Content Delivery Network*)
TLS	Segurança de Camada de Transporte (*Transport Layer Security*)
P2P	*Peer-to-Peer*
TOR	*The Onion Router*
NAT	Tradução de Endereços de Rede (*Network Address Translation*)
IRC	*Internet Relay Chat*
IoT	Internet das Coisas (*Internet of Things*)
ITU-T	União Internacional de Telecomunicações - Setor de Padronização de Telecomunicações
UPnP	*Universal Plug and Play*

Lista de figuras

Lista de tabelas

Sumário

Introdução

A centralização progressiva da Internet tem sido um fenômeno notável ao longo das últimas décadas, a despeito de sua arquitetura fundamentalmente descentralizada. Tal consolidação, causada por diversos fatores econômicos e sociais, fez com que o controle da rede se concentrasse cada vez mais nas mãos de um pequeno conjunto de empresas extremamente influentes. Tal situação levanta grandes preocupações sobre privacidade, segurança, e posse de dados pessoais, num cenário em que poucas aplicações amplamente utilizadas são de fato descentralizadas em arquitetura e governança.

Em uma análise mais detalhada, nota-se que esse fenômeno de centralização tem se apresentado em muitos componentes diferentes. Luciano Zembruzki junto de outros pesquisadores mostrou, por exemplo, como o mercado de hospedagem *web* se tornou extraordinariamente concentrado, e como apenas 10 provedores compunham quase toda a hospedagem dos domínios de topo de nível[1] considerados em sua análise (Zembruzki *et al.*, 2022). Por outro lado, Giovane Moura, Sebastian Castro, e outros em um estudo notaram que o Sistema de Nomes de Domínio tem sido modificado de maneira similar, apresentando "concentração notável" (Moura *et al.*, 2020). Outros artigos relatam comportamentos similares em *Content Delivery Networks* (CDNs) e na propagação da versão mais recente do protocolo de segurança *Transport Layer Security* (TLS) 1.3 (Vermeulen *et al.*, 2023; Holz *et al.*, 2020). Assim, percebe-se que esse processo é muito difundido, e afeta múltiplas facetas técnicas da Internet moderna.

Por outro lado, mesmo numa perspectiva social e econômica a rede mundial de computadores tem passado por um constante processo de consolidação corporativa, fusões entre empresas, e uma presença cada vez maior de tecnologias proprietárias e ecossistemas fechados que evitam a migração dos usuários para serviços concorrentes. Tal processo é descrito em excelente detalhe por Ulrich Dolata no quinto capítulo do livro "Collectivity and Power on the Internet: A Social Perspective" Dolata e Schrape, 2018.

Nesse sentido, a presença cada vez menor de protocolos descentralizados de compartilhamento de arquivos e mensagens é um processo pelo menos parcialmente influenciado por entidades comerciais, uma vez que companhias têm diversas vantagens e incentivos para controlar e manejar cada vez mais dados de seus clientes, ao invés de delegá-los a um

[1] Domínios de topo de nível, como por exemplo .com, .net, .com.br, são a parte final de um nome de domínio, indicando a categoria ou o país de origem do site. Eles são geridos por organizações específicas e são essenciais para a estrutura da Internet. A pesquisa analisou a distribuição desses domínios entre diferentes provedores de hospedagem e concluiu que a maioria deles está concentrada em um pequeno número de empresas, o que pode levar a uma maior vulnerabilidade e controle centralizado da infraestrutura da Internet.

sistema autônomo. Além disso, sistemas abertos e controlados por seus usuários muitas vezes requerem um nível mais elevado de conhecimento técnico para serem utilizados, tornando-se menos preferíveis para o consumidor médio. É muito comum que soluções comerciais tenham mais suporte e sejam mais convenientes e simples de se utilizar, adquirindo assim mais clientes a longo prazo. Assim, um sistema distribuído que consiga lidar com todos os aspectos técnicos da transmissão de mensagens de forma descentralizada pode ser uma solução preferível para um usuário que prefere não ter seus dados pessoais controlados por uma entidade com fins lucrativos.

Neste contexto, esta monografia tem como objetivo apresentar uma análise técnica das limitações de sistemas descentralizados de envio de mensagens, considerando aspectos como a arquitetura, sincronicidade, confiabilidade na entrega e resistência a atores mal-intencionados. Tal análise é usada como motivação para o desenvolvimento de um sistema descentralizado de troca de mensagens que também é apresentado nesta monografia. Embora existam diversas implementações de protocolos descentralizados de código aberto, este trabalho não tem como objetivo criar um protocolo completamente abrangente que substitua todos os demais. Em vez disso, ele busca uma análise crítica e abrangente dessas soluções, destacando suas vantagens e limitações do ponto de vista de implementação. Essa avaliação é fundamental para informar futuros estudos e o desenvolvimento de novos protocolos. Além disso, a criação de uma prova de conceito desempenha um papel essencial neste estudo, ao permitir a aplicação prática das ideias discutidas, demonstrando concretamente seus méritos e desafios.

Desta maneira, este trabalho está dividido em cinco capítulos. O Capítulo 1 apresenta definições relevantes, e explica alguns conceitos importantes para a discussão dos protocolos que serão analisados. O Capítulo 2 analisa implementações já existentes de protocolos de envio de mensagens pela Internet. O Capítulo 3 apresenta a proposta de um protocolo de envio de mensagens descentralizado, e o Capítulo 4 apresenta detalhes sobre a implementação e resultados de testes. A monografia é finalizada com conclusões e ideias para trabalhos futuros no Capítulo 5.

Capítulo 1

Definições importantes

Antes de analisar implementações existentes de protocolos descentralizados de envio de mensagens, é importante definir alguns conceitos e características importantes para a compreensão do funcionamento desses protocolos. Este capítulo será dedicado a explicar tais pré-requisitos.

1.1 Tipos de protocolos de comunicação

Uma primeira distinção importante a ser feita é entre protocolos de comunicação centralizados e descentralizados. A partir dessas definições é possível definir melhor outros termos e conceitos relacionados a protocolos de comunicação.

Definição 1 Um **protocolo centralizado** de envio de mensagens é aquele em que suas comunicações passam por servidores controlados por uma única entidade ou organização. Nesse modelo, os servidores centrais gerenciam a transmissão e roteamento das mensagens. Em muitos casos essa entidade central também pode ser responsável pelo armazenamento, autenticação, e criptografia dos dados dos usuários.

Um protocolo com arquitetura centralizada permite um controle mais rigoroso sobre o fluxo de dados, o que facilita a implementação de funcionalidades como *backups*, moderação de conteúdo e integração com outros serviços. No entanto, essa centralização também implica riscos, como maior vulnerabilidade a falhas de segurança, falhas de serviço e questões relacionadas à privacidade, já que todos os dados dos usuários são concentrados em um único ponto. Tal característica é resumida pelo termo "ponto único de falha". Além disso, a dependência de uma entidade centralizada pode levar a restrições de acesso e à censura, uma vez que o controle das comunicações está nas mãos de uma única organização.

Por outro lado, podemos definir um protocolo descentralizado como:

Definição 2 Um **protocolo descentralizado** de envio de mensagens é aquele em que as comunicações são distribuídas entre vários nós ou servidores, sem a necessidade de uma entidade central para gerenciar ou governar o fluxo de dados. Nesse modelo, os usuários podem se comunicar diretamente uns com os outros, sem depender de intermediários centralizados.

Os protocolos descentralizados têm várias vantagens em relação aos centralizados, como maior resistência a falhas, maior privacidade e menor vulnerabilidade a censura. No entanto, eles também apresentam desafios, como a necessidade de garantir a integridade e a consistência dos dados em um ambiente distribuído e a necessidade de lidar com questões de escalabilidade e desempenho. Além disso, a ausência de uma entidade centralizada pode dificultar a implementação de certas funcionalidades, como *backups*, moderação de conteúdo e integração com outros serviços.

Independentemente do seu nível de centralização ou descentralização, os protocolos de comunicação podem ser classificados de acordo com várias características e funcionalidades. Algumas das características mais importantes incluem:

- **Privacidade de metadados:** Mesmo que a grande maioria dos protocolos modernos de comunicação garantam a privacidade do conteúdo das mensagens, muitos deles ainda coletam e armazenam metadados, como informações sobre quem enviou a mensagem, quando foi enviada e para quem foi enviada. Além disso, durante a sua transmissão, dados relacionados com os dispositivos de origem e destino, informações de interfaces de rede e endereços de IP podem ser coletados por terceiros. Protocolos que protegem a privacidade de metadados são capazes de impedir que essas informações sejam expostas e garantir sua confidencialidade.

- **Escalabilidade:** A escalabilidade é a capacidade de um sistema de crescer e lidar com um aumento no número de usuários e mensagens sem comprometer o desempenho. Protocolos não escaláveis não conseguem lidar com um aumento da demanda mesmo com a adição de mais recursos.

- **Interoperabilidade:** A interoperabilidade é a capacidade de diferentes sistemas e protocolos de comunicação trabalharem juntos de forma eficiente e eficaz. Alguns protocolos analisados nesta monografia são mais versáteis e implementam uma série de padrões e especificações que permitem a comunicação com outros sistemas e protocolos.

- **Assincronicidade:** Protocolos assíncronos permitem que os usuários enviem mensagens sem depender da disponibilidade do destinatário. Esta funcionalidade geralmente depende de um sistema de armazenamento de mensagens, que pode ser centralizado ou não, que permite que as mensagens sejam entregues futuramente quando o destinatário estiver *online*.

1.2 Confidencialidade, integridade e autenticidade

Como mensagens transmitidas pela Internet passam por muitos dispositivos diferentes antes de chegarem ao seu destino, é importante garantir que elas permaneçam seguras e protegidas durante todo o processo. Dentro desse contexto, surgem três definições fundamentais para a segurança de comunicações: confidencialidade, integridade e autenticidade. Stallings em seu livro "Cryptography and Network Security" Stallings, 2017 discute no primeiro capítulo de sua obra as definições desses conceitos, e comenta sobre X.800, uma recomendação da ITU-T que define um modelo de segurança para redes de computadores. Este modelo define a segurança de uma rede em termos de cinco categorias: confidencialidade, integridade, autenticidade, disponibilidade e controle de acesso. Neste trabalho, vamos focar nas três primeiras categorias:

Definição 3 A **confidencialidade** é a proteção dos dados contra divulgação não autorizada.

Definição 4 A **integridade** é a garantia de que os dados recebidos são exatamente os mesmos enviados por uma entidade autorizada (ou seja, não contêm modificações, inserções, deleções ou repetições).

Definição 5 A **autenticidade** é a garantia de que a entidade comunicante é realmente quem ela afirma ser.

Esses três requerimentos podem ser atendidos através de criptografia, que é o processo de codificar informações de forma que apenas pessoas autorizadas possam decodificá-las. Protocolos modernos de criptografia, assinatura digital, e verificação de integridade permitem que qualquer mensagem recebida possa ser verificada, garantindo que veio de um remetente confiável, e que não foi alterada ou lida durante a sua transmissão.

Sistemas de comunicação modernos podem utilizar estas ferramentas em passos discretos do processo de comunicação, ou no caminho completo do remetente ao destinatário. A criptografia de ponta a ponta é um método que garante que os dados sejam criptografados no dispositivo do remetente e só possam ser descriptografados no dispositivo do destinatário. Isso garante que os dados permaneçam seguros, mesmo se forem roteados por uma entidade central, ou sejam transmitidos por uma rede não confiável.

1.3 Redes locais, NATs e *firewalls*

A maioria dos usuários da Internet não possui um endereço de IP público diretamente acessível, o qual é um identificador único e global que permite que um dispositivo seja reconhecido na rede. Em vez disso, eles geralmente estão conectados a uma rede local, que é uma rede privada onde vários dispositivos compartilham um único endereço de IP público. Nessa configuração, o roteador atua como o principal dispositivo de gerenciamento do

tráfego de dados entre a rede local e a Internet, redirecionando as respostas dos serviços externos para os dispositivos corretos na rede. Esse processo realizado pelo roteador é conhecido como *Network Address Translation* (NAT).

Geralmente, dispositivos que estão conectados a uma rede local por meio de um NAT possuem grandes restrições na sua habilidade de receber conexões de fora da rede local. Isso ocorre porque o NAT não sabe para qual dispositivo na rede local deve redirecionar a conexão recebida. Além disso, *firewalls*, que são programas de segurança que monitoram e controlam o tráfego de dados entre a rede local e a Internet, podem bloquear conexões recebidas de fora da rede local. Quando dois dispositivos não conseguem estabelecer uma conexão direta entre si, eles vão precisar de um intermediário para ajudar a encaminhar as mensagens entre eles.

Enquanto muitos roteadores impõe restrições na habilidade de receber conexões externas, algumas técnicas e protocolos podem contornar estar restrições. Uma delas é o *Universal Plug and Play* (UPnP), que é um protocolo que permite que dispositivos na rede local abram portas automaticamente no roteador, permitindo que conexões externas sejam estabelecidas com eles. Outra técnica é o *hole punching*, que é um método que permite que dois dispositivos atrás de NATs diferentes estabeleçam uma conexão direta entre si, sem a necessidade de um intermediário.

1.3.1 *Universal Plug and Play* (UPnP)

O *UPnP* é um protocolo de rede que permite que dispositivos na rede local descubram e se comuniquem uns com os outros de forma automática. Uma das funcionalidades mais importantes do *UPnP* é a capacidade de abrir portas automaticamente no roteador, permitindo que dispositivos na rede local recebam conexões externas. Isso é particularmente útil para aplicações *peer-to-peer* (P2P), que precisam estabelecer conexões diretas entre dispositivos na rede local e dispositivos externos.

O *UPnP* é amplamente utilizado em aplicações de comunicação, como chamadas de voz e vídeo, compartilhamento de arquivos e jogos *online*. No entanto, o *UPnP* também apresenta considerações de segurança, uma vez que permite que dispositivos na rede local abram portas no roteador sem a necessidade de autenticação. Isso pode ser explorado por atacantes para comprometer a segurança da rede local e acessar dispositivos internos. Além disso, nem todos os roteadores suportam o *UPnP*, o que pode limitar a sua eficácia em certas configurações de rede.

1.3.2 *Hole punching*

Hole punching é uma técnica que permite que dois dispositivos atrás de NATs diferentes estabeleçam uma conexão direta entre si, sem a necessidade de um intermediário. A técnica funciona explorando as regras de tradução de endereços do NAT para criar um "buraco"na barreira de segurança do dispositivo. Uma vez que o buraco é criado, os dispositivos podem se comunicar diretamente entre si, sem a necessidade de um intermediário.

Uma consideração importante sobre esta técnica é que ela necessita de um servidor ou outro canal de comunicação entre os dispositivos para coordenar o processo de *hole*

punching. Além disso, a eficácia do *hole punching* pode variar dependendo do tipo de NATs envolvidos, da configuração da rede e de outros fatores. No entanto, quando bem-sucedido, o *hole punching* pode permitir que dispositivos estabeleçam conexões diretas entre si.

1.4 Relays em protocolos de comunicação

Um tipo de dispositivo comumente utilizado em protocolos de comunicação descentralizados é o *relay*.

Definição 6 Um ***relay*** é um servidor intermediário que ajuda a encaminhar mensagens entre diferentes usuários em um protocolo de comunicação descentralizado. Os *relays* geralmente possuem arquitetura interna bem simples, apenas encaminhando mensagens entre usuários, sem armazenar definitivamente ou processar dados.

Relays são principalmente utilizados para contornar restrições de roteamento em redes que possuem camadas de *Network Address Translation* (NAT) ou *firewalls*, que podem impedir a comunicação direta entre dois usuários. Outras funcionalidades que eles podem prover incluem a capacidade de armazenar mensagens temporariamente, para que possam ser entregues mesmo quando o destinatário não está *online*, e a capacidade de diminuir os metadados que estão sendo transmitidos, uma vez que eles podem ocultar o endereço IP real do remetente e destinatário.

1.5 *The Onion Router* (Tor)

O *Tor* é uma rede de comunicação descentralizada que permite que os usuários naveguem na Internet de forma anônima e segura. Ele é composto por uma rede de servidores de voluntários, chamados de *relays*, que se dispõe a redirecionar o tráfego de Internet de forma anônima. Como estes voluntários são desconhecidos, o protocolo da rede é desenhado para minimizar a quantidade de informações que cada *relay* tem acesso, garantindo a privacidade dos usuários. Esta rede pode ser utilizada tanto para navegação em websites comuns da Internet com mais privacidade, assim como para acessar serviços ocultos, que são sites que só podem ser acessados através da rede *Tor*.

1.5.1 Circuitos e troca de chaves

Quando um usuário se conecta à rede *Tor*, o seu tráfego é roteado através de uma série de *relays*, chamada de circuito. Cada circuito é composto usualmente por três *relays*, chamados de entrada, intermediário e saída. O *relay* de entrada é o primeiro a receber a mensagem do usuário, o *relay* intermediário é o segundo a receber a mensagem, e o *relay* de saída é o último a enviar a mensagem para o destino. Cada *relay* apenas tem acesso a informações sobre o *relay* anterior e o próximo, o que ajuda a limitar a quantidade de informações que cada *relay* tem acesso.

O usuário realiza a troca de chaves criptográficas com cada *relay* do circuito, de forma que cada *relay* possa criptografar e descriptografar a mensagem. Quando o usuário vai

enviar um pedido, ele o criptografa com todas as chaves dos *relays* do circuito, começando com a chave do *relay* de saída, depois a chave do *relay* intermediário, e por fim a chave do *relay* de entrada. Cada *relay* descriptografa a mensagem com a sua chave, e analisa se deve encaminhar a mensagem para o próximo *relay* ou atuar como um *relay* de saída.

Esse processo pode ser realizado uma vez para o usuário enviar uma requisição para um website na Internet comum, ou várias vezes para o usuário acessar um serviço oculto.

1.5.2 Acesso a sites comuns

Quando um usuário acessa um site comum da Internet através da rede *Tor*, ele apenas precisa estabelecer um circuito. Ele já é suficiente para obfuscar o tráfego, uma vez que cada *relay* apenas tem acesso a informações sobre o *relay* anterior e o próximo. Isso significa que:

- O *relay* de entrada sabe o endereço IP do usuário, mas não sabe o que ele está acessando.
- O *relay* intermediário não sabe nada sobre remetentes ou destinatários e não pode ver o conteúdo transmitido, apenas sabendo a quantidade de dados transmitidos.
- O *relay* de saída sabe que algum usuário da rede está acessando aquele serviço, mas não sabe quem é o usuário.

Um vídeo excelente publicado pela Universidade de Nottingham que entra em mais detalhes sobre a troca de chaves e a criptografia do protocolo *Tor* pode ser encontrado em University of Nottingham, 2017a.

Uma forma de atestar o funcionamento do *Tor* é por meio do acesso a algum site que verifique a geolocalização do endereço de IP do usuário, como o *mylocation.org*. Esse site utiliza a base de dados de registros de IPs para informar no mínimo em qual país o usuário se encontra com base em seu endereço de IP. A figura 1.1 explica como o site *mylocation.org* determina a localização do *relay* de saída do *Tor*, e não a do usuário. Uma consequência da utlilização do *Tor* é a redução na taxa de transmissão de dados e o aumento da latência nas comunicações. A figura 1.2 mostra como a velocidade de conexão à Internet é reduzida quando o tráfego é roteado através de vários *relays*. Ambas foram capturas de tela do mesmo website, *speedtest.net*, acessado através do *Firefox* e do *Tor Browser*.

A utilização do protocolo *Tor* para acessar sites comuns da Internet é conceitualmente similar a utilizar um serviço de *Virtual Private Network* (*VPN*). Ambos agem como um intermediário entre o usuário e o site acessado, escondendo o endereço IP real do usuário e protegendo a sua privacidade. No entanto, enquanto *Tor* é gratuito, mais anônimo e descentralizado, *VPNs* são na maioria das vezes pagas e centralizadas, mas geralmente possuem melhor desempenho.

1.5.3 Acesso a serviços ocultos

Além de permitir o acesso a sites comuns da Internet de forma anônima, o *Tor* também suporta a criação e acesso a serviços ocultos, que são sites que só podem ser acessados através da rede *Tor*. Estes sites possuem um endereço especial, terminado em .onion, que é gerado a partir de uma chave pública do servidor. A utilização de serviços ocultos permite

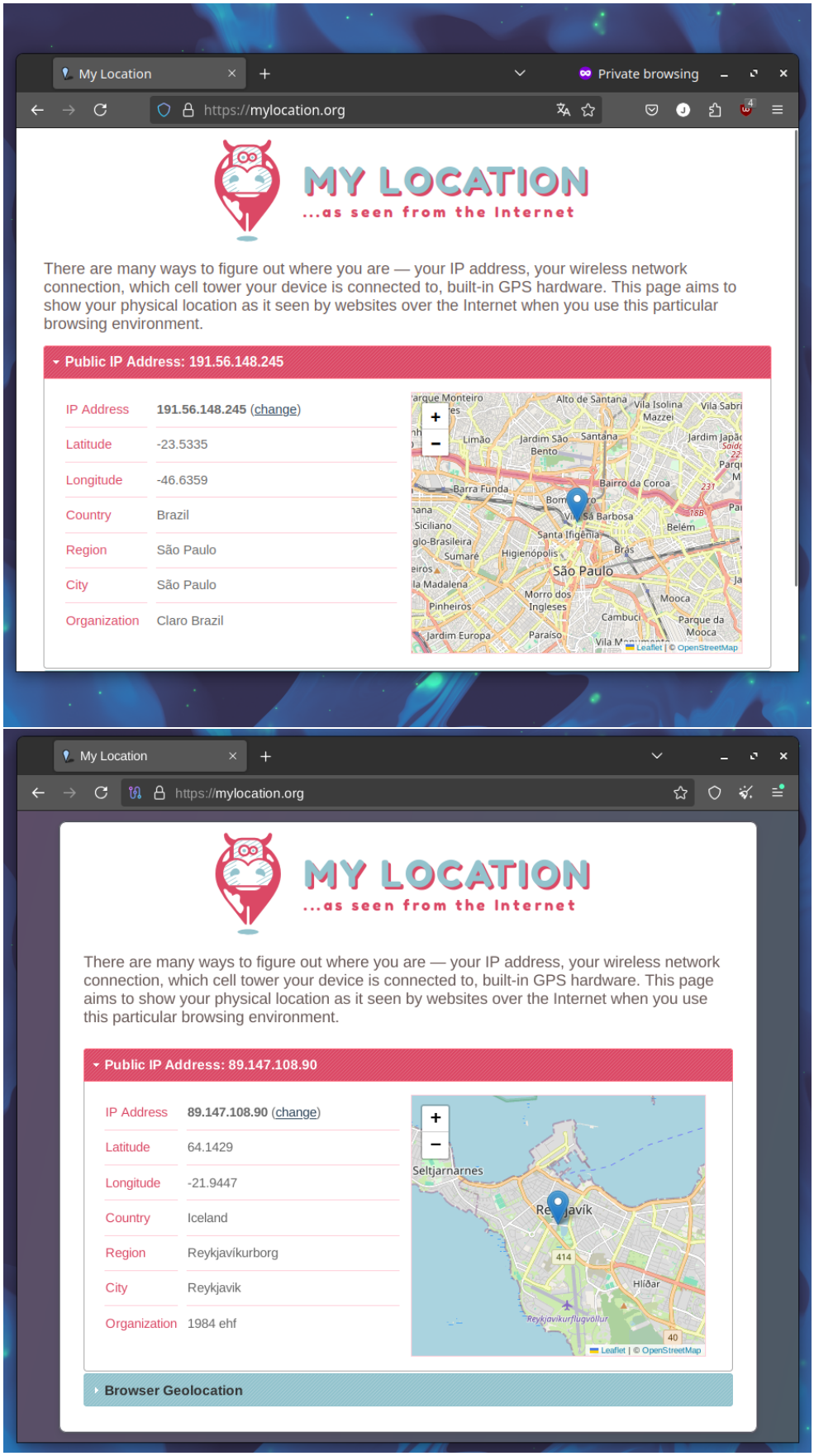

Figura 1.1: *Captura de tela de um website de localização acessado através do Firefox (acima) e do Tor (abaixo). Note que mesmo usando a mesma conexão com a Internet, o website acessado através do Tor determina a localização do Relay de Saída (nesse caso na Islândia), e não a do usuário.*

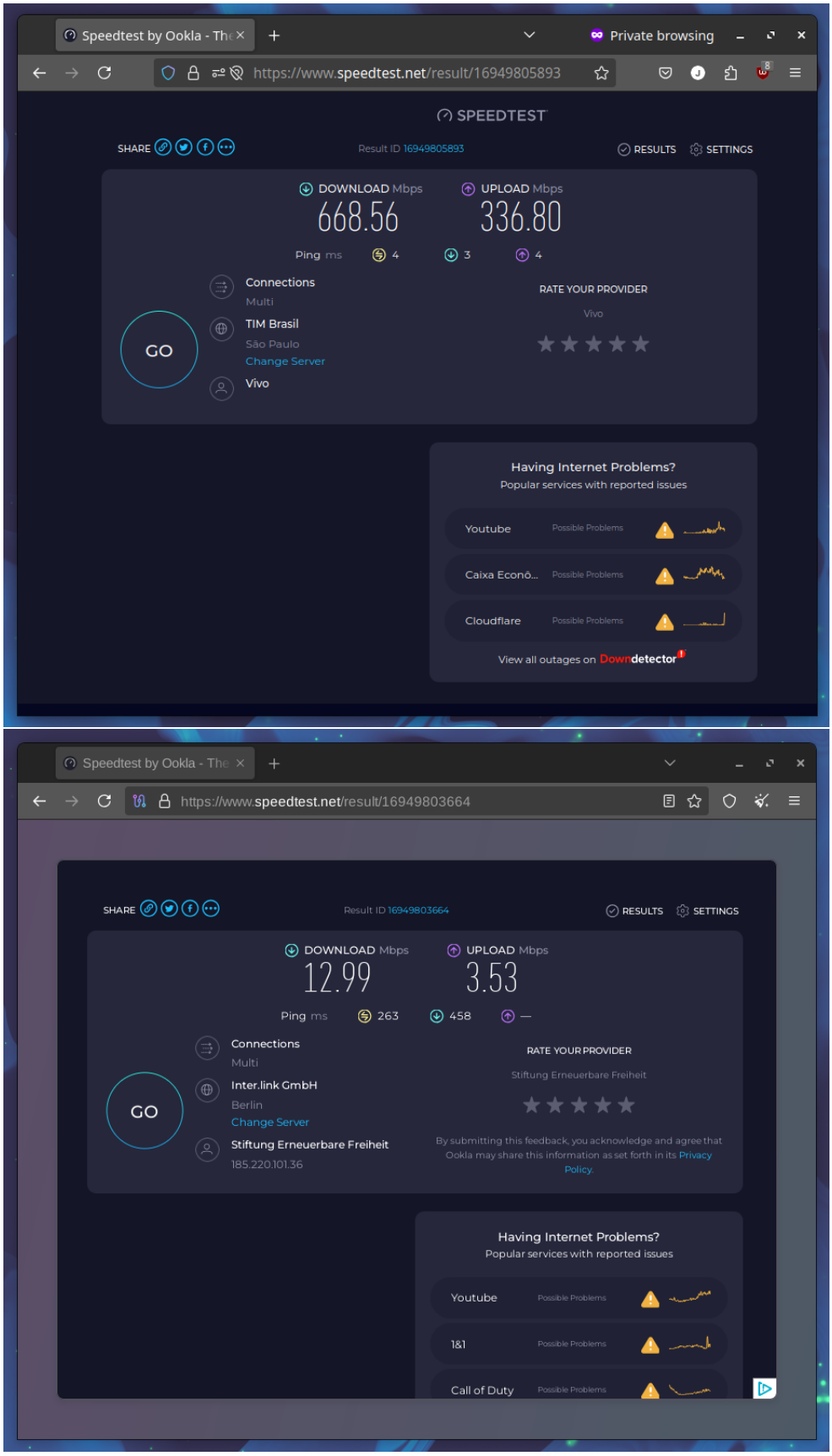

Figura 1.2: *Captura de tela de um de teste de velocidade de conexão à Internet acessado através do Firefox (acima) e do Tor (abaixo). Note que a conexão através do Tor é muito mais lenta, uma vez que o tráfego é roteado através de vários relays, limitando a velocidade da conexão e introduzindo latência.*

que os usuários hospedem sites de forma anônima e segura, contornando restrições de *firewalls* e NATs, sem precisarem expor a localização física do servidor.

Quando um usuário acessa um serviço oculto, ele estabelece uma série de circuitos, buscando que tanto ele quanto o serviço oculto estabeleçam um circuito com um *relay* intermediário em comum. Isso permite que o usuário e o serviço oculto possam se comunicar de forma anônima, sem que nenhum dos dois saiba a localização física do outro.

A Universidade de Nottingham também publicou um vídeo excelente que entra em mais detalhes sobre como funcionam os serviços ocultos do protocolo *Tor*, que pode ser encontrado em University of Nottingham, 2017b.

Serviços ocultos são extremamente úteis no contexto da transmissão segura de mensagens *online*, uma vez que eles lidam com limitações de *firewalls* e NATs, oferecendo comunicação que é segura e anônima. Por outro lado, a grande quantidade de intermediários no encaminhamento de mensagens significa que a latência e a velocidade da comunicação podem ser comprometidas.

Capítulo 2

Implementações já existentes

Este capítulo da monografia realizará a análise de implementações de código aberto já existentes de protocolos descentralizados de envio de mensagens. Essas implementações fornecem uma base para compreender os desafios práticos, as abordagens de arquitetura e as estratégias de mitigação de problemas que foram desenvolvidas e testadas pela comunidade. Os protocolos analisados foram selecionados por serem abertos, descentralizados, e por terem uma arquitetura de rede singular, sendo relevante para este estudo. Os protocolos analisados foram categorizados pela sua arquitetura de rede, e divididos em quatro grupos principais: federados, *peer-to-peer* com *relays* que não usam *Tor*, *peer-to-peer* com *relays* que usam *Tor*, e *peer-to-peer* sem *relays*.

Os protocolos analisados federados são:

- *Matrix*
- *XMPP*
- *E-Mail*

Os protocolos analisados *peer-to-peer* com *relays* que não usam *Tor* são:

- *SimpleX*
- *IRC*
- *LoRaWan*

Os protocolos analisados *peer-to-peer* com *relays* que usam *Tor* são:

- *Cwtch*
- *Ricochet*
- *Briar*

Os protocolos analisados *peer-to-peer* sem *relays* são:

- *Secure Scuttlebutt*
- *Tox*

- *Bitmessage*

A análise de cada protocolo é focada na arquitetura de rede utilizada para troca de mensagens e não considera em detalhes outras funcionalidades, como chamadas de voz, videoconferência ou compartilhamento de arquivos. A seguir, são apresentadas as análises dos protocolos selecionados.

2.1 Protocolos Federados

Na arquitetura federada de envio de mensagens, cada usuário escolhe um servidor para ser seu provedor de mensagens. Essa máquina será responsável por rotear todas as comunicações do usuário e, em alguns casos, também armazenar suas mensagens. É importante que o usuário possa confiar em seu provedor de mensagens, uma vez que ele é um intermediário de todas as comunicações do usuário, que muitas vezes podem não ser criptografadas.

Caso um usuário queira se comunicar com outro usuário em um servidor diferente, os servidores precisam se comunicar entre si para rotear as mensagens. Como o servidor é responsável pelo recebimento e entrega das mensagens, os protocolos federados permitem o envio assíncrono de mensagens. Em geral, os endereços dos usuários nesses sistemas consistem em uma combinação de um nome de usuário com o endereço do servidor.

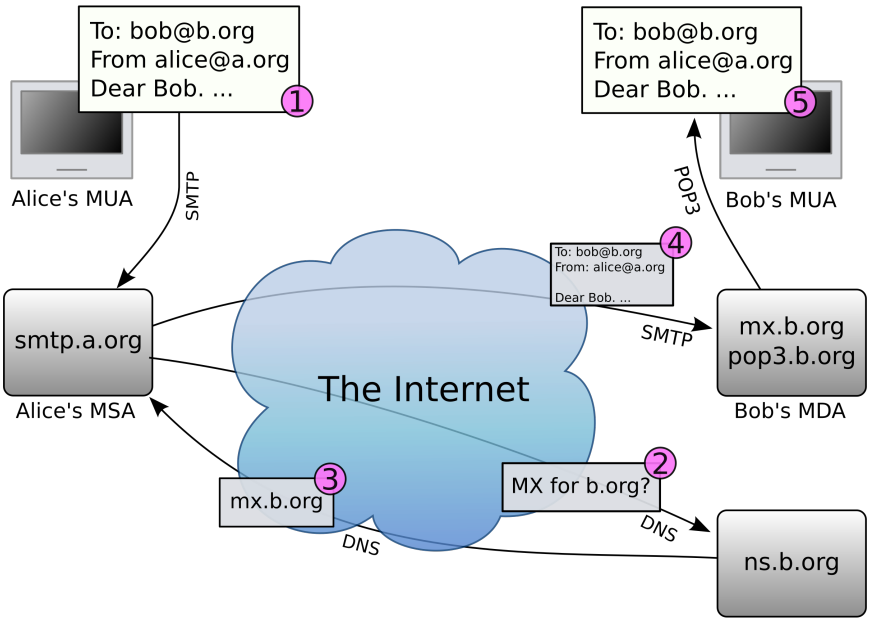

Figura 2.1: *Arquitetura do email, uma aplicação federada, que consiste em cada usuário escolher um servidor para ser seu provedor de mensagens. Mensagens entre usuários de servidores diferentes são roteadas pelos próprios servidores.* COMMONS, 2017

2.1.1 *E-Mail*

O *e-mail* (correio eletrônico) é um sistema de comunicação digital que permite o envio e recebimento de mensagens e arquivos entre usuários através de redes de computadores, utilizando protocolos como *SMTP*, *POP3* e *IMAP* para a transmissão e armazenamento das mensagens. É amplamente utilizado para comunicação pessoal e profissional, sendo uma das formas mais comuns de troca de informações na Internet. Internet Engineering Task Force, 2008

A figura 2.1 ilustra a troca de mensagens via *e-mail* entre o usuário de endereço alice@a.org e o usuário de endereço bob@b.org. Os servidor de a.org determina o endereço do servidor responsável por b.org através do sistema de resolução de nomes (DNS), e envia a mensagem para o servidor de b.org, que por sua vez entrega a mensagem para o usuário bob.

- **Tipo de endereço**: usuário@servidor
- **Suporte a criptografia de ponta a ponta**: Sim

2.1.2 *Matrix*

O protocolo *Matrix* é um sistema de comunicação descentralizado e federado projetado para suportar mensagens instantâneas e colaboração em tempo real. Desenvolvido para ser uma solução aberta e interoperável, o *Matrix* visa fornecer uma infraestrutura robusta para comunicação em diversos contextos, desde *chat* e chamadas de voz até a troca de arquivos. Matrix.org, 2024

A figura 2.2 ilustra a troca de mensagens via *Matrix* entre usuários de diferentes servidores. Cada usuário escolhe um servidor para ser seu provedor de mensagens, e mensagens entre usuários de servidores diferentes são roteadas pelos próprios servidores.

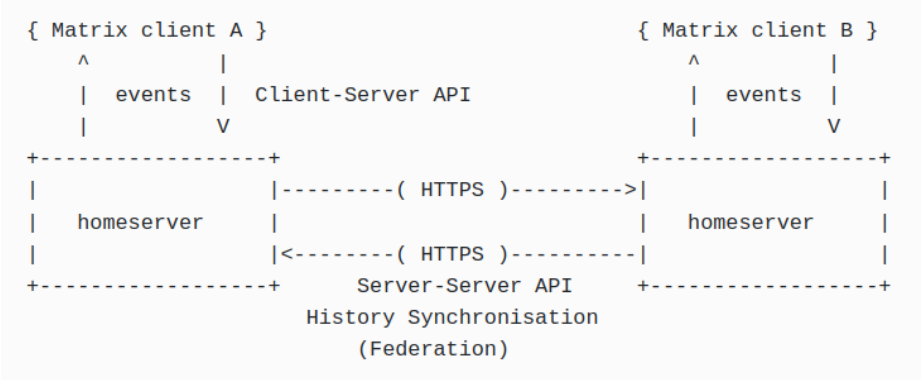

Figura 2.2: *Arquitetura do protocolo Matrix, que assim como o e-mail consiste em cada usuário escolher um servidor para ser seu provedor de mensagens. Mensagens entre usuários de servidores diferentes são roteadas pelos próprios servidores.* Matrix.org, 2024

- **Tipo de endereço**: @usuário:servidor
- **Suporte a criptografia de ponta a ponta**: Sim

2.1.3 *XMPP*

XMPP (*Extensible Messaging and Presence Protocol*) é um protocolo de comunicação em tempo real, baseado em *XML*, que permite a troca descentralizada e segura de mensagens, presença e dados estruturados entre clientes e servidores, suportando uma ampla variedade de funcionalidades, como *chat*, voz, vídeo e colaboração em grupo, sendo reconhecido por sua extensibilidade e interoperabilidade. XMPP Standards Foundation, 2024

- **Tipo de endereço**: usuário@servidor
- **Suporte a criptografia de ponta a ponta**: Sim

2.2 Protocolos Peer-to-Peer com *Relays* que não usam *Tor*

Na arquitetura *peer-to-peer* com *relays*, seja com *Tor* ou não, os usuários se conectam diretamente uns aos outros e armazenam localmente suas mensagens. Para contornar limitações de rede, como camadas de *NAT* e *firewalls*, os usuários podem usar *relays* para intermediar a comunicação. Os *relays* são apenas responsáveis por encaminhar diretamente as mensagens, apagando-as quando elas são entregues. Em alguns protocolos, esses *relays* podem armazenar a mensagem por um curto período de tempo, até o destinatário estar disponível para recebê-la. Os seguintes protocolos utilizam *relays* não participantes na rede do *Tor* para intermediar a comunicação entre usuários.

2.2.1 *SimpleX*

O *SimpleX* é um protocolo de mensagens instantâneas *peer-to-peer* que utiliza *relays* para intermediar a comunicação entre usuários. Sua característica distinta é a sua completa ausência de um identificador de usuário, de modo que usuários podem iniciar uma conversa utilizando um simples código de convite. Simplex Chat, 2024

- **Tipo de endereço**: Código de convite
- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Sim

2.2.2 *IRC*

O *IRC* (*Internet Relay Chat*) é um protocolo de comunicação em tempo real que foi muito popular no final dos anos 90 e início dos anos 2000, e que ainda é utilizado por algumas comunidades *online*. Neste protocolo, mensagens só podem ser recebidas de forma síncrona, uma vez que o servidor se comporta como um *broadcast*, enviando as mensagens para todos os usuários conectados. Internet Engineering Task Force, 2000

- **Tipo de endereço**: endereço de servidor
- **Suporte a criptografia de ponta a ponta**: A maioria dos clientes não suporta
- **Assíncrono**: Não

2.2.3 *LoRaWan*

O *LoRaWan* é um protocolo de comunicação de longo alcance e baixa potência que permite a comunicação entre dispositivos de *Internet of Things* (*IoT*). O protocolo se utiliza tanto de *LoRa*, que é uma tecnologia de modulação de rádio, quanto de redes de *relays* para transmitir mensagens entre dispositivos pela Internet. The Things Network, 2024

- **Tipo de endereço**: endereço do dispositivo
- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Sim

2.3 Protocolos Peer-to-Peer com *Relays* que usam *Tor*

Os seguintes protocolos utilizam a rede do *Tor* para mediar a negociação de chaves e a comunicação entre usuários.

2.3.1 *Cwtch*

O *Cwtch* é um protocolo de mensagens instantâneas *peer-to-peer* que utiliza a rede *Tor* para intermediar a comunicação entre usuários. Cada usuário roda um serviço oculto no *Tor*, e os usuários se conectam diretamente uns aos outros através da rede *Tor*. Cwtch, 2024

- **Tipo de endereço**: Endereço do serviço oculto
- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Não

2.3.2 *Ricochet*

O *Ricochet* funciona de maneira muito similar ao *Cwtch*. Cada usuário também roda um serviço oculto, e conecta-se aos serviços ocultos dos outros usuários. Ricochet, 2024

- **Tipo de endereço**: Endereço do serviço oculto
- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Não

2.3.3 *Briar*

O *Briar* é um protocolo muito versátil, e que permite o envio de mensagens através de *Bluetooth*, *Wi-Fi*, ou da Internet. Quando as mensagens são enviadas pela Internet, ele utiliza a rede *Tor* para intermediar a comunicação entre usuários, e é capaz de funcionar mesmo em redes censuradas. Briar, 2024

- **Tipo de endereço**: Chave pública e privada (por meio de um *QR code*)

- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Sim (com um dispositivo secundário rodando o serviço *Briar Mailbox*)

2.4 Protocolos Peer-to-Peer sem *Relays*

Na arquitetura *peer-to-peer* sem *relays*, os usuários se conectam diretamente, ou através de outros usuários, mas o protocolo não tem computadores dedicados para o intermédio de mensagens. Em geral, esses protocolos precisam que o usuário inicialmente saiba o endereço de outros usuários para poder se comunicar com eles, e estes protocolos são mais vulneráveis a restrições na rede, como *NATs* e *firewalls*.

2.4.1 *Secure Scuttlebutt*

O *Secure Scuttlebutt* é um protocolo de rede social descentralizada. Nele, os usuários enviam suas postagens diretamente ou indiretamente para outros usuários, sendo assim resiliente a falhas e limitações de rede, e permitindo que usuários acessem conteúdo de forma assíncrona. Para permitir o envio indireto de mensagens ele implementa uma *Epidemic Broadcast Tree* (João Leitão e Rodrigues, 2007), que replica mensagens de usuários para seus amigos na rede. (Scuttlebutt, 2024)

- **Tipo de endereço**: endereço, porta e chave pública
- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Sim, por meio de replicação de mensagens

2.4.2 *Tox*

O *Tox* é um protocolo de mensagens *peer-to-peer* que permite o envio de mensagens de texto, assim como chamadas de voz e vídeo. Enquanto ele não permite o envio indireto de mensagens, ele implementa "*Bootstrap Nodes*", que são máquinas que auxiliam usuários a encontrar a *Tabela de Hash Distribuída* (*DHT*, do inglês) do *Tox*, que é usada para encontrar outros usuários. The Tox Project, 2024

- **Tipo de endereço**: Chave pública
- **Suporte a criptografia de ponta a ponta**: Sim
- **Assíncrono**: Não

2.4.3 *Bitmessage*

Bitmessage é um protocolo que apresenta similaridades com o *Bitcoin*, uma vez que usuários precisam resolver um desafio de prova de trabalho para enviar mensagens. Nele, mensagens percorrem a rede inteira até chegar ao destinatário. Bitmessage, 2024

- **Tipo de endereço**: Chave pública
- **Suporte a criptografia de ponta a ponta**: Sim

- **Assíncrono**: Sim

Capítulo 3

Projeto de Protocolo

Agora que já definimos conceitos importantes e analisamos as principais implementações de protocolos de mensagens descentralizados, podemos definir um projeto de protocolo que possua parte das características anteriormente definidas. Este capítulo terá como objetivo explicar as decisões tomadas durante o projeto e quais limitações foram consideradas aceitáveis para o desenvolvimento de uma prova de conceito.

3.1 Objetivos para o projeto

Como não é possível cumprir todos os requisitos possíveis ao desenvolver um protocolo descentralizado, vamos precisar encontrar uma boa combinação de funcionalidades. Para este projeto, vamos considerar os seguintes requisitos:

- **Simplicidade na configuração**: O protocolo não deve precisar de configurações avançadas de rede e deve funcionar em qualquer conexão residencial comum.
- **Pouca dependência em voluntários**: O protocolo deve ser razoavelmente resistente a maus atores e deve evitar depender de voluntários para funcionar.
- **Endereço de usuários estável**: O protocolo deve permitir que usuários tenham um endereço único e estável, para que possam ser identificados de maneira fácil.
- **Privacidade**: O protocolo deve evitar expor informações pessoais dos usuários e deve criptografar os dados do usuário quando possível.

Considerando esses requisitos, podemos já definir algumas características da nossa prova de conceito.

3.2 Proposta de arquitetura de rede

Como discutido no primeiro capítulo, as arquiteturas dos protocolos de rede analisados podem ser divididas em quatro grupos diferentes: federados, indiretos por meio de *relays* específicos da rede, indiretos por meio da rede do *Tor* e *peer-to-peer*. Podemos eliminar algumas dessas possibilidades com base nos requisitos que definimos.

Como consideramos que facilidade na configuração é um requisito importante, podemos concluir que não devemos desenvolver um serviço de arquitetura federada. Esse tipo de arquitetura depende de servidores mais centralizados ou que os usuários tenham conhecimento técnico e uma configuração de rede muito específica para que possam hospedar seu próprio servidor. Como buscamos um protocolo que consiga ser configurado sem esforço e que funcione em qualquer configuração de rede, vamos considerar outra arquitetura para esse projeto.

Queremos evitar que o protocolo seja muito dependente de *relays* específicos da rede para funcionar. Esses *relays* dependem de um serviço centralizado para que o usuário os encontre, e eles podem ser facilmente controlados por maus atores, especialmente quando o protocolo não possui muitos usuários. Como queremos que o protocolo não possua essas vulnerabilidades, não vamos desenhar o protocolo com essa arquitetura.

O protocolo do *Tor* é uma opção interessante, pois ele é descentralizado e não depende de servidores específicos para funcionar. Ele possui grande quantidade de *relays* voluntários, que vão continuar existindo independentemente do número de usuários utilizando este protocolo, e é bastante resistente a maus atores. Sua desvantagem é a quantidade de passos necessários para estabelecer uma conexão, o que gera uma latência maior. O *Tor* também soluciona um problema muito grande da arquitetura *peer-to-peer*, que é a dificuldade de encontrar outros usuários na rede, uma vez que endereços de IP residenciais são muitas vezes dinâmicos. Assim, vamos propor para esse projeto um programa de arquitetura híbrida, que utiliza o *Tor* para encontrar outros usuários e uma arquitetura *peer-to-peer* para a comunicação entre eles.

3.2.1 Arquitetura de rede proposta: híbrida

Na arquitetura proposta por este projeto, quando um usuário estiver *online*, ele deve hospedar um serviço oculto com seu par de chaves público-privadas. O serviço oculto de cada usuário irá expor uma *API* para outros usuários. Quando ele quiser realizar qualquer ação com outro usuário, ele deve conectar-se ao serviço oculto do outro usuário e utilizar algum dos *endpoints* disponíveis. Esses *endpoints* permitem diversas funcionalidades, como enviar mensagens, checar se o outro usuário o considera um "amigo"ou até mesmo solicitar uma conexão *P2P* direta.

Consideramos que a privacidade é uma característica importante para o protocolo. Sabemos que a forma de comunicação por serviços ocultos proposta acima tem a grande vantagem de esconder o endereço IP de ambos os usuários, ao custo de maior latência. Enquanto algumas pessoas podem preferir se comunicar de forma anônima, outros usuários podem preferir conversar através de uma conexão direta, que é mais rápida e mais eficiente. Para isso, podemos propor um sistema de "amigos"para o protocolo. Quando um usuário considerar que prefere abrir mão de seu anonimato na conversa com outro usuário, ele pode escolher adicionar aquele endereço como amigo. Quando dois usuários são amigos, eles podem se comunicar diretamente, sem passar pelo serviço oculto. Isso permite que usuários que se conhecem e confiam um no outro possam se comunicar de forma mais eficiente, enquanto ainda mantêm a privacidade de usuários que não se conhecem.

Para evitar que o servidor do programa esteja constantemente monitorando outros

usuários e tentando estabelecer uma conexão *P2P*, podemos propor que toda vez que um usuário abre a conversa com outro, ele solicite uma conexão *P2P*. Se ambos os usuários estiverem *online*, ambos considerarem o outro como amigo, ambos estiverem com o *chat* aberto com a outra pessoa e as configurações de rede permitirem, a conexão *P2P* é estabelecida. Caso contrário, a comunicação é feita através do serviço oculto.

Esta solução permite que mensagens sejam enviadas em qualquer configuração de rede, permite que usuários se comuniquem de forma anônima, permite que os usuários se comuniquem de forma direta, não depende de *relays* específicos da rede e é resistente a maus atores.

3.3 Detalhes da implementação do protocolo

A implementação desse protótipo foi organizada em torno de um único executável que lida com todos os aspectos do servidor. Este executável roda o *Tor* como um subprocesso e sobe todos os *endpoints* necessários para a comunicação entre usuários. Alguns desses *endpoints* são acessíveis apenas localmente, enquanto outros serão acessíveis pela rede *Tor*. A interface do usuário é feita através de um cliente *web*, que se comunica com o executável através de uma *API*.

Dada a experiência do autor dessa monografia com a linguagem de programação *Python*, escolheu-se utilizá-la para o desenvolvimento do protótipo. *Python* é uma linguagem de programação de alto nível, que possui uma grande quantidade de bibliotecas disponíveis para diversas funcionalidades. Além disso, *Python* é uma linguagem de programação muito utilizada para desenvolvimento de aplicações *web*, o que facilita a interação com a interface do usuário. O cliente *web* é escrito em *HTML*, *CSS* e *JavaScript*, e não utiliza nenhum *framework* de *frontend*.

Enquanto o servidor utiliza muitas bibliotecas, a principal é *Flask*. *Flask* é um *framework web* minimalista para *Python*, que permite a criação de aplicações *web* de forma rápida e fácil. *Flask* é muito utilizado para a criação de *APIs* e é muito fácil de se utilizar.

3.3.1 Servidor de *backend*

O servidor de *backend* é composto por 12 arquivos, cada um com funções de um propósito específico.

- **main.py**: Arquivo principal que inicializa o servidor e gerencia subprocessos.
- **__init__.py**: Vazio, utilizado para indicar que a pasta é um pacote *Python*.
- **endpoints.py**: Inicializa todos os *endpoints*, inicializa variáveis, e inicia o servidor de *Flask*.
- **friends.py**: Contém muitas funções relacionadas a amigos, como *requests* sobre amizades, pedidos de endereço de IP, pedidos de conexão *P2P*, e funções relacionadas. Esses pedidos serão melhor explicados mais a frente.
- **jsonOperator.py**: Contém funções para ler e escrever todos os dados que o servidor precisa armazenar. O estado do programa é escrito em um arquivo *json*. As funções

de escrita utilizam um *lock* para evitar condições de corrida.

- **middleware.py**: Tanto o tráfego do *Tor* como o tráfego do cliente chegam à mesma *API*. Para diferenciar entre os dois, utilizamos um *middleware* que adiciona um *header* "Tor-Middleware-Header", indicando se o *request* veio do *Tor* ou do cliente.
- **p2p.py**: Contém funções relacionadas à conexão direta entre participantes, seja por meio do *localhost*, rede local, pela internet utilizando *UPnP*, ou por meio de *relays* do *Tor*. É o arquivo mais complexo do servidor.
- **privateEndpoints.py**: Contém *endpoints* que não devem ser acessíveis pela rede *Tor*, como por exemplo de acesso às mensagens armazenadas, para envio de mensagens, ou para adicionar amigos.
- **publicEndpoints.py**: Contém *endpoints* que devem ser acessíveis pela rede *Tor*, como por exemplo para solicitar endereços de IP, solicitar conexões *P2P*, ou para enviar mensagens.
- **serverCrypto.py**: Contém funções relacionadas à criptografia, como por exemplo assinaturas eletrônicas, criptografia de mensagens, e geração de chaves.
- **setup.py**: Contém funções para inicializar o *Tor*, configurando portas, arquivos de configuração, e inicializando o processo.
- **upnp.py**: Contém funções para abrir portas no roteador do usuário, utilizando o protocolo *UPnP*. Contém uma implementação própria porque a biblioteca de *UPnP* para *Python* não funcionou corretamente.

O servidor abre uma série de *endpoints* locais e públicos, que são acessíveis pela rede *Tor*. Os *endpoints* públicos são:

- **pubEndpoint_receiveMessage (POST)**: *Endpoint* utilizado para receber mensagens de outros usuários. As mensagens são criptografadas de ponta a ponta pelo próprio protocolo do *Tor* e são armazenadas no servidor.
- **pubEndpoint_getPublicKeyBase64 (GET)**: *Endpoint* que retorna a chave pública do usuário em formato *Base64*. Essa chave é utilizada para criptografar mensagens enviadas para o usuário e para verificar assinaturas digitais.
- **pubEndpoint_checkFriendRequest (POST)**: *Endpoint* que verifica se há um pedido de amizade pendente de um determinado usuário.
- **pubEndpoint_getIpRequest (POST)**: *Endpoint* que solicita o endereço IP de um usuário específico. Utilizado para tentar estabelecer uma conexão *P2P*.
- **pubEndpoint_getFriendIP (POST)**: *Endpoint* que retorna o endereço IP de um amigo, caso ambos os usuários tenham se adicionado como amigos.
- **pubEndpoint_p2pRequest (GET)**: *Endpoint* que solicita o estabelecimento de uma conexão *P2P* direta entre dois usuários.
- **pubEndpoint_receiveGenericFriendRequest (POST)**: *Endpoint* utilizado para receber pedidos genéricos de amizade de outros usuários.

- **pubEndpoint_ping (GET)**: *Endpoint* utilizado para verificar se o servidor está *online* e respondendo às requisições.

Os *endpoints* locais são:

- **privEndpoint_getMessagesFromSender (POST)**: *Endpoint* utilizado para obter mensagens de um remetente específico.
- **privEndpoint_sendMessage (POST)**: *Endpoint* utilizado para enviar mensagens para outros usuários.
- **privEndpoint_getLatestMessages (GET)**: *Endpoint* utilizado para obter as mensagens mais recentes.
- **privEndpoint_getSenders (GET)**: *Endpoint* utilizado para obter a lista de remetentes.
- **privEndpoint_getAddress (GET)**: *Endpoint* utilizado para obter o endereço do usuário.
- **privEndpoint_startChat (POST)**: *Endpoint* utilizado para iniciar uma conversa com outro usuário.
- **privEndpoint_getFriends (GET)**: *Endpoint* utilizado para obter a lista de amigos.
- **privEndpoint_addFriend (POST)**: *Endpoint* utilizado para adicionar um amigo.
- **privEndpoint_removeFriend (POST)**: *Endpoint* utilizado para remover um amigo.
- **privEndpoint_changeFocusedFriend (POST)**: *Endpoint* utilizado para mudar o amigo focado na interface.
- **privEndpoint_getFriendConnectionStatus (GET)**: *Endpoint* utilizado para obter o status de conexão de um amigo.

3.3.2 Cliente *web*

O cliente *web* é composto por uma aba de conversas e uma aba de amigos. A aba de conversas mostra uma lista com todas as pessoas com as quais o usuário já conversou e permite que o usuário selecione uma conversa para visualizar. A aba de amigos mostra uma lista com todos os amigos do usuário e permite que o usuário adicione ou remova amigos.

Como o cliente não é o escopo principal deste projeto, ele é simples e apenas cumpre os requisitos necessários ao projeto. As imagens 3.1, 3.2 e 3.3 mostram respectivamente a conversa entre dois usuários, a aba que contém os amigos do usuário e a aba principal da interface.

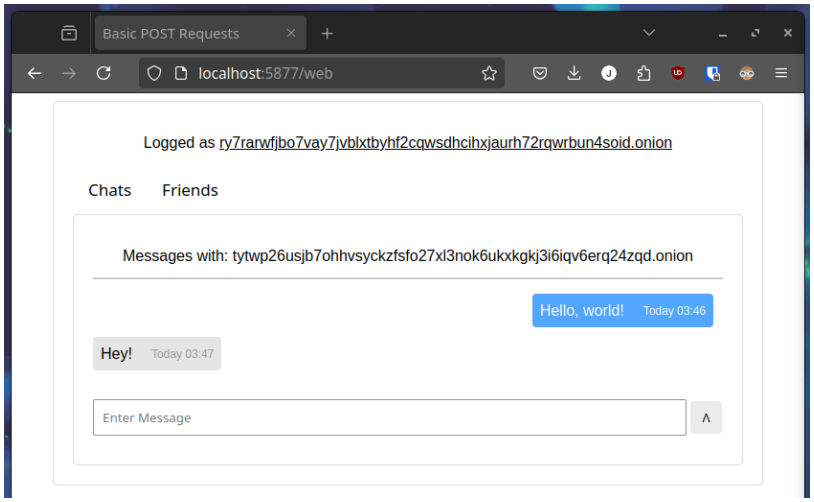

Figura 3.1: *Captura de tela da interface mostrando a conversa entre dois usuários.*

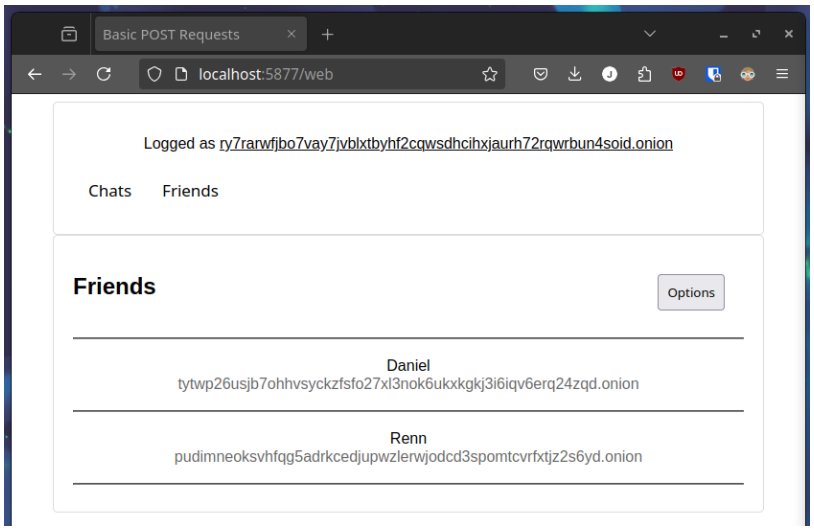

Figura 3.2: *Captura de tela da aba que contém os amigos de um dado usuário.*

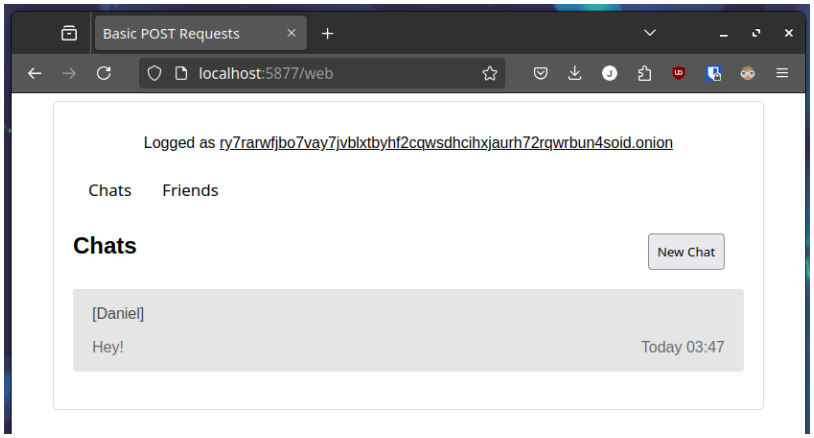

Figura 3.3: *Captura de tela da aba principal da interface, contendo todas as conversas de um dado usuário.*

3.4 Criptografia e *handshakes*

O servidor utiliza alguns métodos de criptografia para garantir a segurança das mensagens trocadas entre os usuários. A intenção inicial do projeto era que o servidor utilizasse a mesma chave pública e privada que o subprocesso do *Tor*, porém, como o *Tor* usa regras de derivação de chave diferentes das da biblioteca *ED25519*[1] do *Python*, os servidores vão compartilhar chaves privadas, mas vão utilizar chaves públicas diferentes para acessar o *Tor* e para assinar e criptografar mensagens. Isto poderia gerar um problema de verificação de autenticidade da chave pública, uma vez que o protocolo do *Tor* garante a autenticidade de uma chave pública realizando a derivação do endereço a partir da chave pública, permitindo que, ao acessar um serviço oculto, o usuário tenha certeza de que a chave pública que ele recebeu está correta. Assim, o primeiro contato entre dois servidores teve que ser repensado para garantir que ambos os usuários consigam assegurar que têm a chave pública correta de seus amigos.

Sabemos que, ao acessar um serviço oculto, o *Tor* realiza a verificação da chave pública recebida. Assim, sabemos que, quando queremos iniciar uma conversa com um usuário novo, podemos simplesmente acessar o *endpoint pubEndpoint_getPublicKeyBase64* do usuário, e o *Tor* irá garantir que a chave pública que recebemos é a correta. Porém, como garantir que a chave pública que recebemos é a correta quando recebemos uma mensagem de um usuário com o qual nunca falamos antes? Para resolver esse problema, propomos um *handshake* inicial entre os dois servidores.

Quando o servidor receber um pedido de envio de mensagem de um usuário com o qual ele nunca falou antes, ele irá responder a esse pedido de envio de mensagem com a mensagem "refaça esse envio, vou pedir sua chave pública primeiro". O servidor então irá acessar o *endpoint pubEndpoint_getPublicKeyBase64* da pessoa que está enviando a mensagem, e assim que receber esse *request* novamente, irá verificar se a chave pública recebida é a mesma que a chave pública que ele recebeu anteriormente. Esse processo acontece de forma síncrona, de modo que o segundo pedido de envio de mensagem demorará um pouco para ser respondido, sendo completado quando o servidor tiver certeza de que a chave pública recebida é a correta.

Uma vez que ambos os servidores têm chaves públicas com autenticidade confirmada, eles podem assinar mensagens, criptografar mensagens e verificar assinaturas digitais.

A criptografia de mensagens não é necessária quando os usuários estão conversando através do *Tor*, uma vez que o protocolo do *Tor* já criptografa as mensagens de ponta a ponta. Porém, quando os usuários estão conversando de forma direta, a criptografia é necessária.

3.4.1 Pedidos exclusivos para amigos

Alguns pedidos entre servidores requerem uma "prova"de autenticidade. Por exemplo, caso o usuário A queira pedir o endereço IP do usuário B, ele deve provar que é amigo de

[1] ED25519 é um esquema de assinatura digital e criptografia de chave pública baseado no algoritmo de curva elíptica Edwards. Ele permite que mensagens sejam assinadas e sua autenticidade seja verificada, e permite que usuários criptografem mensagens para outros usuários, apenas tendo acesso a chave pública do destinatário.

B. Pedidos como esse têm um formato padronizado e são gerados e verificados no arquivo *friends.py*. O servidor que recebe o pedido verifica se o pedido é válido e, se for, responde com a informação solicitada. O formato de cada pedido é o seguinte:

- **origin**: Endereço do usuário que está fazendo o pedido.
- **destination**: Endereço do usuário que está recebendo o pedido.
- **timestamp**: Hora exata em que o pedido foi feito.
- **kind**: Tipo de pedido que está sendo feito.
- **signature**: Assinatura digital do pedido, gerada a partir do *JSON* do pedido.

Dessa maneira, pedidos não podem ser reutilizados e pedidos não podem ser feitos em nome de outro usuário.

3.5 *Handshake P2P*

Quando um usuário abre uma conversa na interface, o servidor vai seguir uma série de passos para determinar se uma conexão direta entre usuários pode ser feita.

1. O servidor verifica se os usuários são amigos mútuos.
2. O servidor espera que o outro usuário também tenha esta conversa aberta.
3. O servidor pede os endereços IP do outro usuário.
4. Se os usuários tiverem endereços IP públicos e privados iguais, ele tenta estabelecer uma conexão pelo *localhost*.
5. Se os usuários tiverem endereços IP públicos iguais e endereços IP privados diferentes, ele tenta estabelecer uma conexão pela rede local.
6. Se os usuários tiverem endereços IP públicos diferentes, ele tenta estabelecer uma conexão direta entre eles usando *UPnP*.

Caso qualquer um desses passos não seja bem-sucedido, os usuários ainda poderão comunicar-se normalmente usando o *Tor* como intermediário.

3.6 Considerações finais

A implementação do protocolo não é um produto finalizado e não deve ser utilizada como um substituto para programas de envio de mensagens popularmente utilizados. Embora ele consiga cumprir os requisitos que foram propostos, ele não foi testado extensivamente no mundo real e provavelmente é vulnerável a ataques. Ainda assim, ele é uma demonstração muito interessante de um tipo de arquitetura que não foi muito explorado por projetos acadêmicos até agora. O protocolo foi implementado como software livre seguindo a licença *GNU GPL v3* e está disponível publicamente em https://linux.ime.usp.br/~renner/MAC0499/ (hospedado pela infraestrutura da USP) e https://github.com/LeRenner/mac0499-prototype (hospedado por um serviço externo). O

código deste programa contém dependências disponíveis por meio de outras licenças. O arquivo /LICENÇA.txt contém uma lista de todas as dependências e suas licenças.

Capítulo 4

Análise de desempenho

A implementação do protótipo foi testada em diversas conexões com a *internet*, tanto institucionais, como a rede da Universidade de São Paulo, quanto residenciais e móveis. Como o *UPnP* não é universalmente compatível com todos os roteadores, a maior parte dos testes que utilizaram *UPnP* foi realizada em conexões residenciais de uma empresa que tem roteadores compatíveis com o protocolo. Como a rede do *Tor* não é restrita no Brasil, em nenhuma das conexões testadas houve problemas de acesso aos serviços ocultos de outros usuários. Os resultados dos experimentos realizados para avaliar o desempenho da proposta apresentam algumas considerações sobre as limitações deste protótipo.

4.1 Tempo de inicialização do *Tor*

O tempo de inicialização do *Tor* é uma inconveniência para a mobilidade deste protocolo. Enquanto esse programa termina de se inicializar em menos de 10 segundos na maioria dos casos, os serviços ocultos que são hospedados por ele demoram mais para serem adicionados à Tabela de *Hash* Distribuída do protocolo. Quando o *Tor* tenta acessar um serviço oculto que ainda não terminou de ser inicializado, ele demora muito para desistir da tentativa de conexão, gerando uma má experiência ao usuário. Por causa disso, é importante que o usuário espere pelo menos um minuto depois da inicialização completa do servidor para tentar se comunicar com outros usuários.

Uma vez que a conectividade pelo *Tor* é essencial ao protocolo, mesmo em casos em que os usuários estão se comunicando de forma *P2P*, é importante que o usuário tenha uma conexão estável com a rede do *Tor*. Isso torna este protótipo inconveniente e imprático para utilização em movimento, como em um *smartphone*, por exemplo.

4.2 Limitações do *UPnP*

O *UPnP* é um protocolo que não é universalmente compatível com todos os roteadores. Em alguns casos, o *UPnP* não é habilitado por padrão, e em outros casos, o roteador não é compatível com o protocolo. Enquanto seria possível implementar instruções para o usuário manualmente abrir portas em seu roteador, isto é uma inconveniência e não foi

implementado neste protótipo. Quando a conexão *P2P* não é possível, como por exemplo nesse caso, o programa simplesmente continua se comunicando com outros usuários através do *Tor*.

Inicialmente, o programa utilizaria a biblioteca *miniupnpc* para abrir portas automaticamente, porém, ao longo dos testes, ela apresentou problemas relacionados com o *timeout* de tentativas de conexão com o roteador. Esta biblioteca, ao tentar descobrir dispositivos compatíveis com *UPnP* na rede, ficava indefinidamente esperando por uma resposta, tornando o processo de descoberta de dispositivos *UPnP* extremamente demorado. Por causa disso, o arquivo *upnp.py* apresenta uma implementação própria que utiliza o executável *upnpc*, que não sofre com este problema específico. Por outro lado, o executável *upnpc* também apresentava problemas de inconsistência na resposta de pedidos. Pedidos para a abertura de portas específicas entravam na tabela de roteamento, mas não estavam ativos. Além disso, muitos comandos terminavam com códigos de erro, mesmo tendo sido bem-sucedidos. Assim, a implementação teve que manualmente testar se os comandos executados eram bem-sucedidos, limpando todos os roteamentos pedidos quando encontrava um erro.

4.3 Velocidade de conexões e latência

Como a implementação prática do protocolo apenas permite o envio de mensagens de texto, que ocupam muito pouco espaço em disco, a velocidade da conexão entre dois usuários é em grande parte irrelevante. O parâmetro mais importante que foi analisado foi a latência de envio de mensagens entre os dispositivos. Para cada ação entre dispositivos foram coletadas 6 medidas de latência, e a média dessas medidas foi utilizada como a latência da conexão. As Tabelas 4.1 e 4.2 apresentam os resultados dessas medições. Como mencionado anteriormente, como o *Tor* demora para tornar os serviços ocultos acessíveis externamente, os programas foram iniciados e os testes só foram conduzidos depois de 1 minuto.

Ação	Tempo Médio	Desvio Padrão
Pedido da chave pública do destinatário	5.94s	0.83s
Handshake inicial entre dois clientes	13.82s	1.77s

Tabela 4.1: *Medições de latência para pedido da chave pública e handshake*

Ação	Tempo Médio	Desvio Padrão
Envio de mensagens (*Tor*)	1740ms	220ms
Envio de mensagens (*UPnP*)	5.83ms	0.75ms
Envio de mensagens (Rede local)	2.33ms	0.52ms
Envio de mensagens (*localhost*)	8ms	1.41ms

Tabela 4.2: *Medições de latência para envio de mensagens*

As máquinas utilizadas para estes testes contém as seguintes especificações técnicas:

Máquina 1:

- Servidor montado para hospedar websites e armazenar aquivos em uma conexão residencial
- Processador: AMD Ryzen 5 1600 AF
- Memória: 16GB DDR4 2666MHz ECC
- Armazenamento: SSD NVMe 256GB
- Sistema Operacional: Debian Linux 12 Bookworm
- Conexão com a *Internet*: Conexão residencial de fibra ótica com 600MBits/s (Download) e 300MBits/s (Upload)

Máquina 2:

- Laptop para uso pessoal (Thinkpad L14 Gen 3)
- Processador: AMD Ryzen 5 5500U
- Memória: 16GB DDR4 3200MHz
- Armazenamento: SSD NVMe 256GB
- Sistema Operacional: Manjaro Linux 24.2.0 (Gnome)
- Conexão com a internet: Mesma que a Maquina 1 (Testes de rede local) através de cabo de rede, e conexão com a rede Wi-Fi Eduroam da USP (Testes de rede externa)

O envio de mensagens através do *Tor* passa as mensagens por 6 computadores da rede do *Tor* antes que a mensagem seja entregue, tornando este método de conexão o mais limitado em latência. Uma consideração interessante é que, por causa do *handshake* inicial do *Tor*, a primeira conexão é muito mais demorada do que as demais. Uma vez que ambos os servidores já se comunicaram, seu caminho fica armazenado, tornando conexões posteriores muito mais rápidas. Por causa disso, o pedido da chave pública (que é o primeiro passo no envio da primeira mensagem entre dois usuários) e o *handshake* (que é o primeiro pedido no sentido contrário do que a mensagem será enviada) são muito mais lentos do que o envio de mensagens subsequentes.

De forma muito interessante, o envio de mensagens através de *UPnP* e rede local é mais rápido do que o envio de mensagens através de *localhost*. Isso se deve ao fato de que mensagens enviadas pelo *localhost* utilizam a biblioteca de *Python Requests*, que é complexa e mais lenta, enquanto pedidos enviados pela rede local e *UPnP* são enviados diretamente para o *socket*, sem intermediários. O envio de mensagens através de *UPnP* é mais lento do que o envio de mensagens pela rede local por causa da latência da comunicação através da *internet*. De qualquer maneira, nos três casos, o envio das mensagens é extremamente rápido, apenas demorando alguns milissegundos.

Os resultados desses testes foram realizados em situações bem ideais de rede, com conexões cabeadas e provedores de *internet* de alta qualidade. Condições adversas de rede definitivamente alterarão esses resultados.

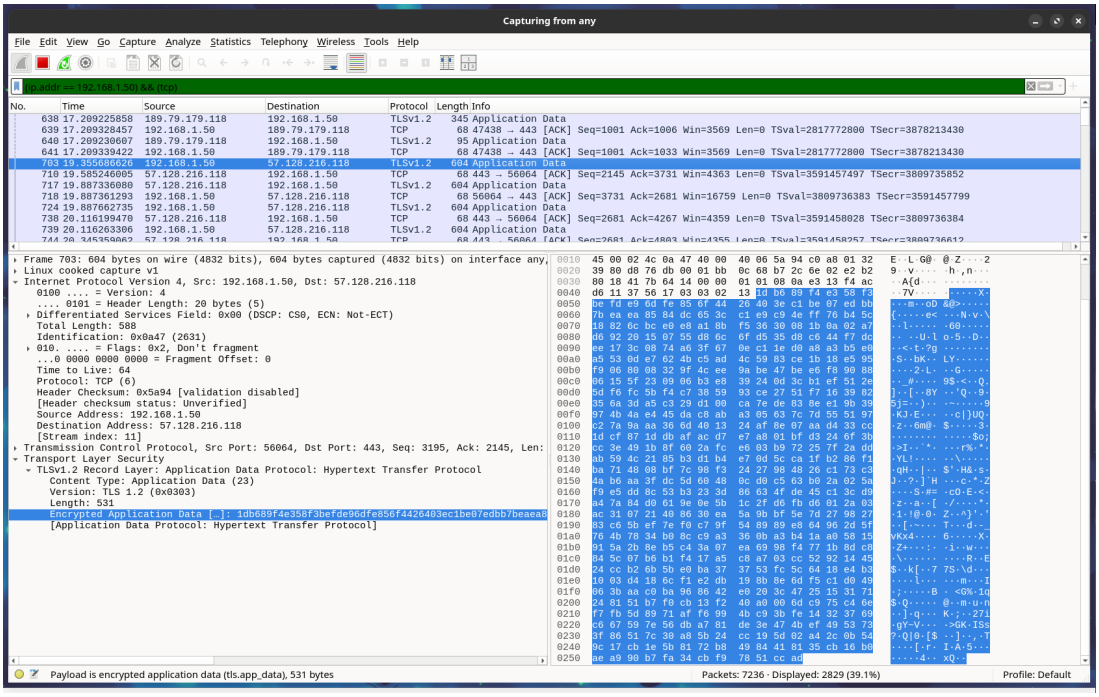

Figura 4.1: *Captura de tela do Wireshark mostrando um pacote criptografado pelo Tor. Note que o conteúdo do pacote é criptografado, e aparece portanto como uma série de caracteres aleatórios. A barra mais a direita, sublinhada em azul, contém o conteúdo do pacote em hexadecimal e sendo interpretado como texto.*

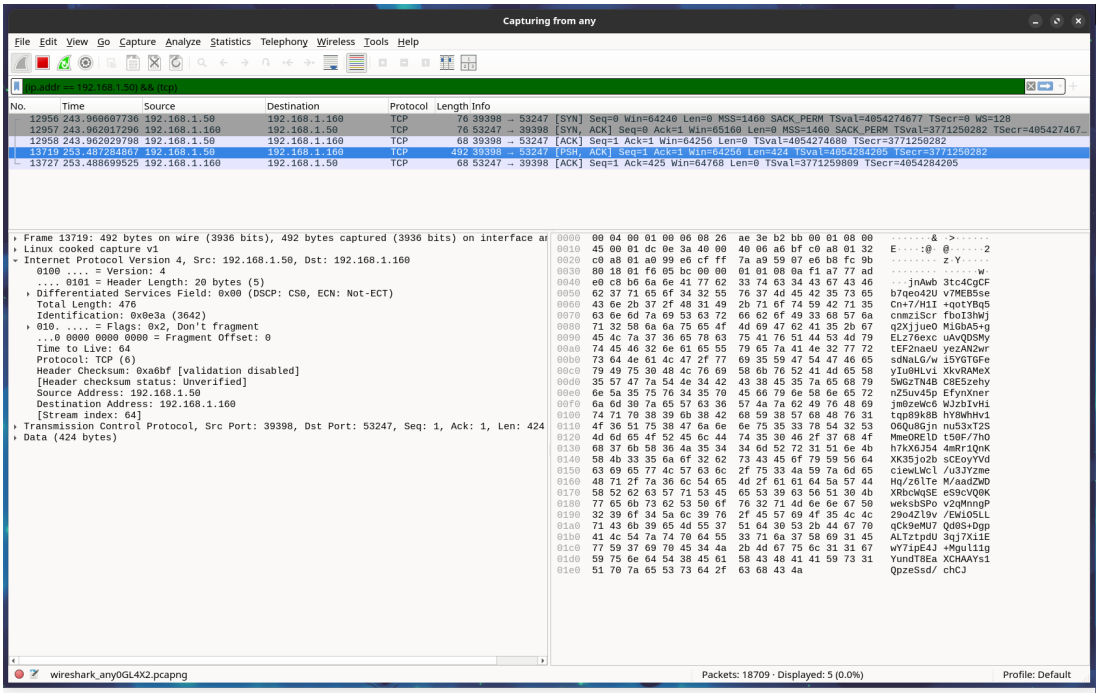

Figura 4.2: *Captura de tela do Wireshark mostrando um pacote criptografado pelo protótipo em uma conexão entre amigos em uma rede local. O conteúdo do pacote é texto criptografado, e depois codificado em base64. Por isso, o conteúdo do pacote é ilegível a uma pessoa não autorizada.*

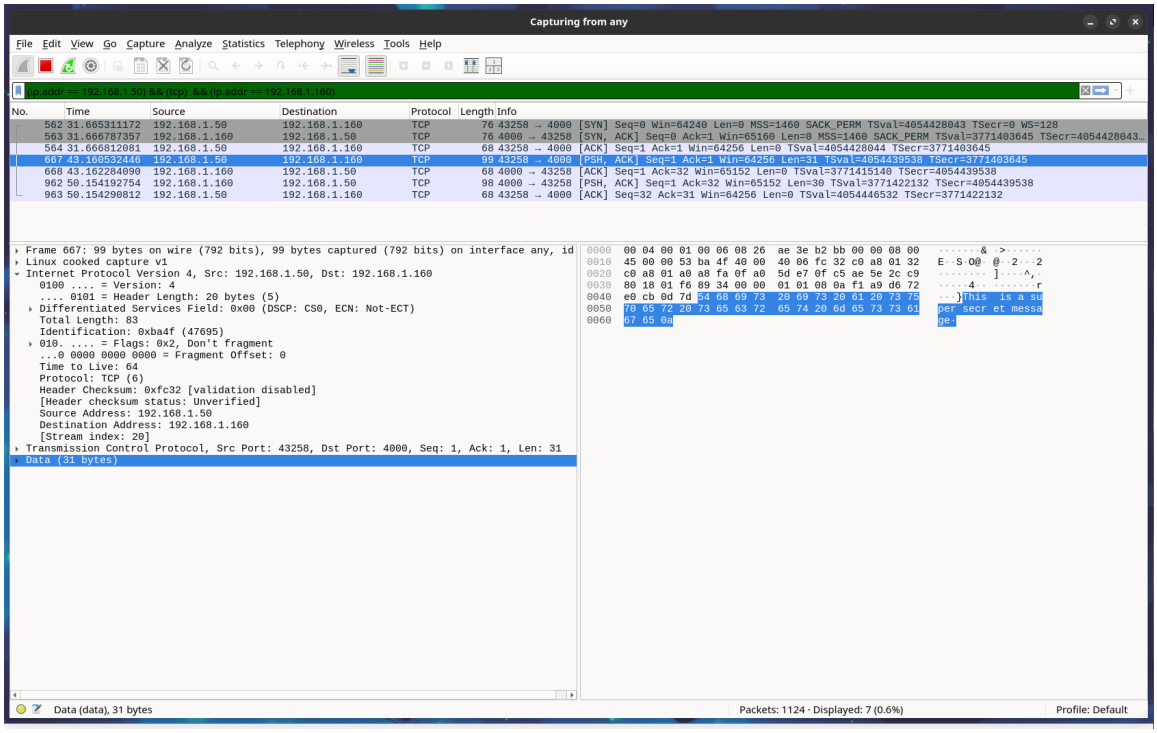

Figura 4.3: *Captura de tela do Wireshark mostrando um pacote de dados não criptografado como exemplo. Neste caso, é possível ver o conteúdo do pacote em texto, na direita da tela.*

As Imagens 4.1, 4.2, e 4.3 mostram capturas de tela do programa Wireshark, que é um programa de análise de pacotes de rede. Ele permite analisar tráfego que é acessível pelas interfaces de rede da máquina em que ele roda. Neste exemplo, ele é utilizado para demonstrar como o tráfego gerado pelo protótipo não pode ser decodificado por atores mal intencionados. A Imagem 4.1 mostra um pacote criptografado pelo *Tor*, que é ilegível a uma pessoa não autorizada. A Imagem 4.2 mostra um pacote criptografado pelo protótipo em uma conexão entre amigos em uma rede local, que é codificado em base64 e também ilegível a uma pessoa não autorizada. A Imagem 4.3 mostra um pacote qualquer de dados não criptografados, que é legível a qualquer pessoa que tenha acesso ao pacote.

Capítulo 5

Conclusão e Trabalhos Futuros

O protótipo que foi desenvolvido mostra como um protocolo híbrido de comunicação, utilizando tanto a rede do *Tor* quanto conexões diretas entre usuários, pode permitir a comunicação entre usuários sem a atuação de nenhuma entidade intermediária controlando o fluxo de mensagens. Todavia, essa implementação é limitada por problemas de latência e estabilidade de conexão, que são inerentes ao uso do *Tor*. Além disso, enquanto este protocolo não depende de voluntários dedicados a este específico protocolo, ele faz uso de uma infraestrutura de rede que é mantida por voluntários. Modificações no protocolo, mudanças na legislação, ou até mesmo uma diminuição no uso e popularidade do *Tor* podem tornar este protocolo inutilizável.

Além disso, a implementação de *UPnP* utilizada não é ideal e pode ser considerada mais próxima de um improviso do que uma implementação de produção. A biblioteca *miniupnpc*, que foi utilizada inicialmente, apresentou problemas de *timeout* e de inconsistência nas respostas dos roteadores. A implementação atual, que utiliza o executável *upnpc*, é mais estável, mas ainda apresenta problemas de inconsistência nas respostas dos roteadores. Enquanto o *UPnP* é um protocolo muito interessante e com muito potencial em implementações de protocolos descentralizados e *P2P*, é necessário que este protocolo tenha mais adoção e que as implementações sejam mais estáveis e consistentes.

A implementação de um protocolo de comunicação *P2P* é um desafio técnico, e a implementação de um protocolo de comunicação *P2P* que seja seguro e anônimo é ainda mais desafiadora. Este protótipo é uma prova de conceito que mostra que é possível implementar um protocolo de comunicação *P2P* que seja seguro e anônimo, mas que ainda tem muitas limitações práticas.

5.1 Trabalhos Futuros

O protótipo implementado não abrange completamente todas as configurações possíveis de rede. Um excelente exemplo que não foi abrangido nesta implementação, por exemplo, é um computador que tenha um endereço de IP público em sua interface de rede. Neste caso, o programa não precisaria de *UPnP* para abrir portas no roteador e poderia se comunicar diretamente com outros usuários. Além disso, a implementação

atual não permite a comunicação direta entre usuários que estão atrás de *NATs* simétricos, que são *NATs* que não permitem a comunicação entre dois usuários que estão atrás de *NATs* diferentes. Uma implementação futura poderia utilizar técnicas diferentes, como *hole punching*, para permitir a comunicação entre usuários que estão atrás de *NATs* simétricos.

Mesmo que a maioria dos programas de envio de mensagens popularmente usados sejam centralizados, como o *WhatsApp*, o *Telegram* e o *Facebook Messenger*, a exploração de protocolos descentralizados e *P2P* é extremamente interessante em um cenário de cada vez maior centralização de poder na *internet*. Grandes plataformas *online* já tiveram momentos de instabilidade, e situações como essas permitem que plataformas descentralizadas sejam mais atrativas para usuários que buscam controle e privacidade de seus dados pessoais.

Referências

[Bitmessage 2024] Bitmessage. *Bitmessage*. Acesso em 14 de agosto de 2024. 2024. URL: https://github.com/Bitmessage/PyBitmessage (citado na pg. 18).

[Briar 2024] Briar. *Briar*. Acesso em 14 de agosto de 2024. 2024. URL: https://briarproject.org/how-it-works/ (citado na pg. 17).

[Commons 2017] Wikimedia Commons. *Email.svg*. Acesso em 22 de agosto de 2024. 2017. URL: https://commons.wikimedia.org/wiki/File:Email.svg (citado na pg. 14).

[Cwtch 2024] Cwtch. *Cwtch*. Acesso em 14 de agosto de 2024. 2024. URL: https://docs.cwtch.im/ (citado na pg. 17).

[Dolata e Schrape 2018] Ulrich Dolata e Jan-Felix Schrape. *Collectivity and Power on the Internet: A Sociological Perspective*. Springer, 2018. ISBN: 978-3319784137 (citado na pg. 1).

[Holz *et al.* 2020] Ralph Holz *et al.* "Tracking the deployment of tls 1.3 on the web: a story of experimentation and centralization". *ACM SIGCOMM Computer Communication Review* 50.3 (jul. de 2020), pp. 3–15. ISSN: 0146-4833. DOI: 10.1145/3411740.3411742. URL: http://dx.doi.org/10.1145/3411740.3411742 (citado na pg. 1).

[Internet Engineering Task Force 2000] Internet Engineering Task Force. *Internet Relay Chat: Architecture*. Acesso em 14 de agosto de 2024. 2000. URL: https://datatracker.ietf.org/doc/html/rfc2810 (citado na pg. 16).

[Internet Engineering Task Force 2008] Internet Engineering Task Force. *Simple Mail Transfer Protocol*. Acesso em 14 de agosto de 2024. 2008. URL: https://datatracker.ietf.org/doc/html/rfc5321 (citado na pg. 15).

[João Leitão e Rodrigues 2007] José Pereira João Leitão e Luís Rodrigues. *Epidemic Broadcast Trees*. Acesso em 26 de agosto de 2024. 2007. URL: https://www.dpss.inesc-id.pt/~ler/reports/srds07.pdf (citado na pg. 18).

[Matrix.org 2024] Matrix.org. *Matrix Specification*. Acesso em 14 de agosto de 2024. 2024. URL: https://spec.matrix.org/latest/ (citado na pg. 15).

[Moura *et al.* 2020] Giovane C. M. Moura, Sebastian Castro, Wes Hardaker, Maarten Wullink e Cristian Hesselman. "Clouding up the internet: how centralized is dns traffic becoming?" In: *Proceedings of the ACM Internet Measurement Conference*. IMC '20. ACM, out. de 2020. doi: 10.1145/3419394.3423625. url: http://dx.doi.org/10.1145/3419394.3423625 (citado na pg. 1).

[Ricochet 2024] Ricochet. *Ricochet*. Acesso em 14 de agosto de 2024. 2024. url: https://github.com/ricochet-im/ricochet (citado na pg. 17).

[Scuttlebutt 2024] Scuttlebutt. *Scuttlebutt*. Acesso em 14 de agosto de 2024. 2024. url: https://ssbc.github.io/scuttlebutt-protocol-guide/ (citado na pg. 18).

[Simplex Chat 2024] Simplex Chat. *Simplex Chat*. Acesso em 14 de agosto de 2024. 2024. url: https://github.com/simplex-chat/simplex-chat (citado na pg. 16).

[Stallings 2017] William Stallings. *Cryptography and Network Security: Principles and Practice*. 7th. Pearson, 2017. isbn: 978-0133354690 (citado na pg. 5).

[The Things Network 2024] The Things Network. *LoRaWAN*. Acesso em 14 de agosto de 2024. 2024. url: https://www.thethingsnetwork.org/docs/lorawan/ (citado na pg. 17).

[The Tox Project 2024] The Tox Project. *Tox*. Acesso em 14 de agosto de 2024. 2024. url: https://github.com/TokTok/c-toxcore (citado na pg. 18).

[University of Nottingham 2017a] University of Nottingham. *How TOR Works*. Acesso em 15 de agosto de 2024. 2017. url: https://www.youtube.com/watch?v=QRYzre4bf7I (citado na pg. 8).

[University of Nottingham 2017b] University of Nottingham. *TOR Hidden Services*. Acesso em 15 de agosto de 2024. 2017. url: https://www.youtube.com/watch?v=lVcbq_a5N9I (citado na pg. 11).

[Vermeulen *et al.* 2023] Kevin Vermeulen, Loqman Salamatian, Sang Hoon Kim, Matt Calder e Ethan Katz-Bassett. "The central problem with distributed content: common cdn deployments centralize traffic in a risky way". In: *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks*. HotNets '23. ACM, nov. de 2023. doi: 10.1145/3626111.3628213. url: http://dx.doi.org/10.1145/3626111.3628213 (citado na pg. 1).

[XMPP Standards Foundation 2024] XMPP Standards Foundation. *XMPP: Core*. Acesso em 14 de agosto de 2024. 2024. url: https://xmpp.org/about/technology-overview/ (citado na pg. 16).

[Zembruzki *et al.* 2022] Luciano Zembruzki, Raffaele Sommese, Lisandro Zambenedetti Granville, Arthur Selle Jacobs e Giovane C. M. Moura. "Hosting industry centralization and consolidation". In: *2022 IEEE/IFIP Network Operations and Management Symposium*. NOMS 2022. IEEE, 2022. URL: https://doi.org/10.48550/arXiv.2109.01187 (citado na pg. 1).