# Uma investigação sobre tolerância a falhas no servidor de comunicação Synapse

**Vinicius Stocker**[1], **Edson Tavares de Camargo**[1], **Luis C. E. De Bona**[2]

[1]Universidade Tecnológica Federal do Paraná, Campus Toledo.
Rua Cristo rei, 19 - Vila Becker - CEP 85902-490 - Toledo, PR - Brasil

[2]Universidade Federal do Paraná, Departamento de Informática.
Rua Cel. Francisco H. dos Santos, 100 - Jardim das Américas -
CEP 81531-980 - Curitiba, PR - Brasil

{viniciusstocker,edson}@utfpr.edu.br.br, bona@inf.ufpr.edu.br

***Abstract.*** *Synapse is an open-source communication server designed according to the Matrix communication protocol. The Matrix protocol is decentralized, federated, and open source, providing complete control of the messaging server by the organization. Messages exchanged between users are end-to-end encrypted, and the protocol can also interact with other proprietary messaging systems and technologies, acting as a middleware. However, a fundamental issue for the widespread adoption of the protocol is its ability to continue its services even in the face of failures. In this sense, the objective of this work is to investigate, through a practical experiment, the ability of the Synapse server to tolerate failures and synchronize lost messages after their recovery.*

***Resumo.*** *Synapse é um servidor de comunicação de código aberto projetado de acordo com o protocolo de comunicação Matrix. O protocolo Matrix é descentralizado, federado e de código aberto, proporcionando controle total do servidor de mensagens pela organização. As mensagens trocadas entre usuários são criptografadas de ponta a ponta e o protocolo ainda pode interagir com outros sistemas de mensagens proprietários e tecnologias, atuando como um middleware. No entanto, uma questão fundamental para a ampla adoção do protocolo é a sua capacidade de continuar seus serviços mesmo diante de falhas. Nesse sentido, o objetivo deste trabalho é investigar, através de um experimento prático, a capacidade do servidor Synapse de tolerar falhas e de sincronizar as mensagens perdidas após sua recuperação.*

## 1. Introdução

Os meios de comunicação modernos, enraizados no ambiente digital, têm desempenhado um papel fundamental na maneira como a sociedade compartilha informações desde o surgimento da *Internet*. Nesse contexto, a confiabilidade, a integridade e a disponibilidade da própria informação e da rota que ela percorre entre o emissor e o receptor, tornaram-se aspectos críticos nas conversações e análises sobre o tema [Sule et al. 2021, Ausat and Almaududi 2023, Cao et al. 2024].

Entre as numerosas opções de aplicativos e protocolos para troca de mensagens, despontam o *WhatsApp* e o Telegram, impulsionados pela sua popularidade e pela expressiva quantidade de usuários ativos. No entanto, ambos são sistemas fechados que não oferecem ao usuário controle total sobre seus dados, exigindo confiança na ética das empresas

para o tratamento das informações processadas em seus servidores. Mesmo os metadados trocados podem trazer informações sensíveis que podem ser exploradas visando comprometer a privacidade dos usuários [Domenech et al. 2022, Nelson et al. 2024]. Contudo, há situações em que a informação trocada entre as partes não pode ser confiada a empresas terceirizadas e, em alguns casos, nem mesmo permitir que as mensagens e dados dos usuários saiam dos servidores da própria organização.

Nestes cenários desafiadores, as plataformas auto-hospedadas como Zulip [Zulip 2025], Chatwoot [Chatwoot 2025] e Element [Element 2025] emergem como possível alternativa. Todas as plataformas citadas apresentam como elemento crucial a capacidade do usuário manter seu próprio servidor de comunicação. Dessa forma, há a certeza de que suas mensagens não necessitam transitar por servidores de empresas terceiras. A plataforma Element é composta do cliente, que recebe o mesmo nome da plataforma, e pelo servidor, chamado Synapse. A comunicação entre eles é construída com base no protocolo Matrix [Matrix 2025a] de comunicação e, diferente das demais, apresenta a capacidade de criar redes federadas entre seus servidores e distribuir a carga de mensagens somente entre os participantes das conversações.

Matrix é um padrão aberto para comunicação interoperável, descentralizada e em tempo real sobre IP (*Internet Protocol*), implementando serviços de assinatura e recebimento de mensagens com base em tópicos de uma rede federada de servidores [Jacob et al. 2019]. O protocolo pode ser usado para alimentar mensagens instantâneas, sinalização VoIP (*Voice Over IP*)/WebRTC (*Web Real-Time Communications*) e comunicação no contexto de Internet das Coisas (ou *Internet of Things* - IoT) [Matrix 2025b]. Há ainda a interoperabilidade com outros aplicativos de conversação e protocolos por meio da instalação de *bridges*. No caso de conversas em uma sala privada, as mensagens trocadas somente deixarão o servidor do usuário perante autorização para participar da sala de conversação. Esta abordagem descentralizada promove uma maior autonomia e controle sobre os dados, ao mesmo tempo em que possibilita a interconexão e a troca de informações de forma eficiente entre diferentes entidades e organizações.

O interesse no protocolo Matrix é crescente entre governos, empresas e academia. O Governo Francês criou a plataforma Tchap com o objetivo de oferecer uma solução de comunicação segura e confiável para as instituições governamentais e seus funcionários [Tchap 2025]. O Ministério da Defesa Alemão criou o BWMessenger para o departamento de defesa e o expandiu para todas as Forças Armadas alemãs [Heise 2019]. Houve ainda a implantação do Matrix no sistema de saúde alemão [Pätsch 2021]. Já o governo de Luxemburgo implementou a plataforma de comunicação Luxchat para toda a população e a plataforma Luxchat4Gov, exclusiva para os funcionários do governo [Karhu 2023].

No contexto empresarial, a Fundação Mozilla desenvolveu o sistema chamado Mozilla Discourse para substituir seus sistemas de comunicação interna anteriores [Discourse 2019]. Já o aplicativo Matrix-CRDT [El-Dardiry 2022] é um editor de arquivos colaborativo e distribuído que utiliza o protocolo Matrix como *backend*, ou seja, é um exemplo da capacidade do protocolo de não ser útil somente para a troca de mensagens de texto. No meio acadêmico, trabalhos como [Jacob et al. 2021], [Schipper et al. 2021], [Ihle et al. 2022] e [Albrecht et al. 2023] avaliam diferentes aspectos do protocolo que vão desde sua escalabilidade até questões de segurança da informação, como análise forense e autenticação descentralizada. No entanto, uma avaliação sobre a capacidade do

protocolo Matrix de tolerar falhas não foi encontrada na literatura.

Um sistema tolerante a falhas pode ser definido como aquele que possui a capacidade de continuar funcionando corretamente mesmo que alguns de seus componentes falhem [Avizienis et al. 2004]. Essa capacidade é um pilar da dependabilidade (*dependability)*, definida como confiança no funcionamento do sistema. Um dos atributos mais importantes de um sistema confiável é a disponibilidade (*availability*). A disponibilidade se refere à capacidade do sistema de estar disponível para uso, levando em conta a ocorrência de falhas que interrompam o serviço e a sua recuperação. A eficácia do protocolo Matrix em contextos específicos é diretamente influenciada por sua resiliência a falhas.

Originalmente voltado para comunicação entre humanos, o Matrix emerge como uma solução poderosa de *middleware* para a Internet das Coisas, permitindo a inclusão de um maior número de entidades na comunicação e suportando um volume e frequência aumentados de mensagens. Suas funcionalidades, como a formação de grupos de comunicação e mecanismos de presença, são particularmente vantajosas para aplicações IoT. A integração com o protocolo XMPP (*Extensible Messaging and Presence Protocol*) [Al-Masri et al. 2020] comum em implementações IoT, amplia ainda mais sua aplicabilidade. Em aplicações críticas, como o monitoramento de saúde, ou em usos governamentais, a continuidade do serviço, garantida pela tolerância a falhas, é fundamental para a adoção em larga escala do Matrix [Pastório et al. 2020].

O objetivo deste trabalho é investigar o comportamento do servidor Synapse, que hoje é a principal implementação [Matrix 2025a] do Matrix, perante falhas do tipo parada (*crash)*. Nesse tipo de falha, o processo que a sofre não responde a qualquer estímulo. A proposta é entender até que ponto esse *downtime* do servidor de um usuário prejudica a sua comunicação e se o protocolo consegue recuperar as mensagens que deixaram de ser entregues a esse usuário quando o servidor retorna ao sistema. Para isso, uma instalação do Synapse com múltiplos servidores e usuários é realizada com a consequente inserção de falhas. Resultados mostram que o servidor não possui meios de garantir alta disponibilidade, embora consiga recuperar as mensagens do usuário caso a sua chave criptográfica não seja perdida devido à falha.

O restante deste artigo está organizado da seguinte forma: A seção 2 detalha a plataforma Matrix/Synapse. A seção 3 apresenta aplicações onde a alta disponibilidade do protocolo é requerida. A seção 4 apresenta o cenário de avaliação e, por fim, a seção 5 descreve as conclusões.

## 2. Descrição do protocolo Matrix e do servidor Synapse

O Matrix é um protocolo de código aberto para comunicação em tempo real ou seja, é uma estrutura versátil de troca de mensagens, adaptável para uso por pessoas, dispositivos IoT, sinalização VoIP/WebRTC ou qualquer outra aplicação capaz de expressar a troca de mensagens por meio de arquivos JSON (*JavaScript Object Notation*) [Sparber 2020]. Sua arquitetura se fundamenta no conceito de salas de conversação, que podem abrigar um ou vários usuários trocando mensagens. Cada um desses usuários deve estar vinculado a um *homeserver*, que é o representante do usuário na rede federada, e somente os servidores contendo usuários participantes da sala em questão efetuarão a troca de mensagens, não sobrecarregando toda a rede federada [Matrix 2025a]. Uma representação de como a rede

federada trabalha pode ser vista na Figura 1.

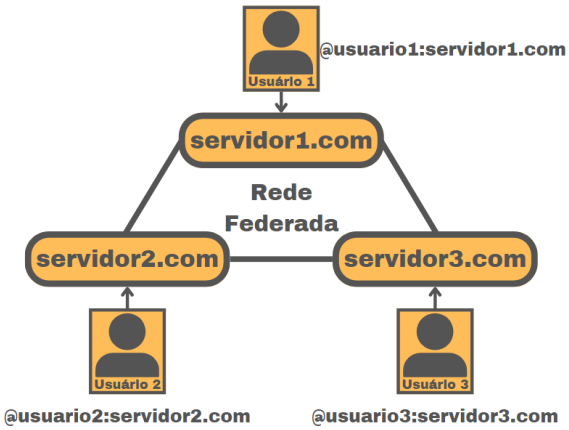


**Figura 1. Rede federada**

Em implementações recentes, o servidor Matrix/Synapse adota a utilização de *workers*, conforme a Figura 2, que são processos separados para executar tarefas específicas em paralelo com o processo principal do servidor. Isso é feito para melhorar o desempenho e a escalabilidade do servidor. Os *workers* são essencialmente processos adicionais que lidam com tarefas como processamento de mensagens, gerenciamento de conexões de rede e operações de manutenção. Distribuir essas tarefas entre múltiplos *workers* permite que o servidor lide com uma carga de trabalho maior de forma mais eficiente, garantindo uma experiência de conversação mais estável e responsiva para os usuários, especialmente em ambientes com alto volume de tráfego. A comunicação entre o processo principal do servidor e os *workers* geralmente é feita de forma assíncrona, permitindo uma coordenação eficaz das tarefas e garantindo que o servidor possa escalar horizontalmente conforme necessário.

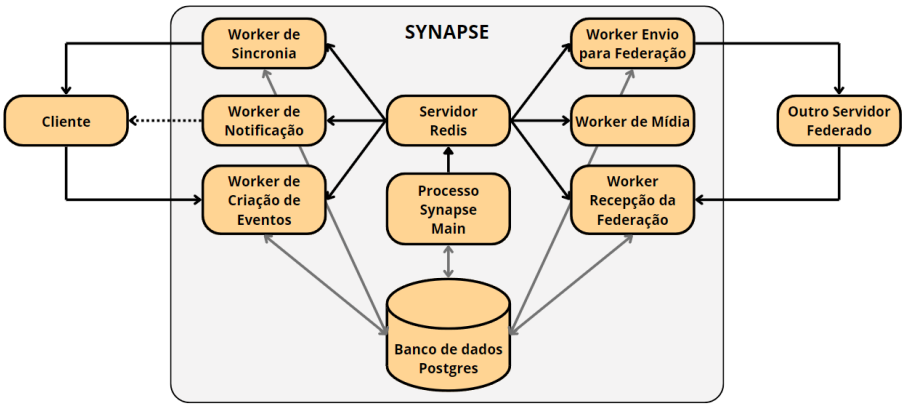


**Figura 2. Implementação de workers no servidor Synapse.**

## 2.1. Matrix e a estrutura de dados Conflict-free Replicated Data type

A robustez do protocolo na troca de mensagens entre os participantes da conversação se baseia no conceito de Grafos de Eventos ou *Matrix Event Graph* (MEG), que foi estruturado para ser um Tipo de Dado Replicado Livre de Conflito ou *Conflict-Free Replicated Data Type* (CRDT), conforme mostrado por [Jacob et al. 2021]. Um CRDT

[Preguiça et al. 2019] fornece um modelo de consistência forte eventual ou *Strong Eventual Consistency* (SEC). Basicamente, a SEC garante que quaisquer dois nós que tenham recebido o mesmo conjunto (não ordenado) de atualizações estarão no mesmo estado [Kleppmann 2022].

Um CRDT é um tipo de dados projetado para suportar operações em sistemas distribuídos, onde múltiplas cópias dos dados podem existir em diferentes nós da rede e precisam ser sincronizadas. A principal característica dos CRDTs é sua capacidade de garantir a convergência dos dados, independentemente da ordem em que as operações são executadas ou da latência da comunicação entre os nós. Isso significa que, mesmo que ocorram atualizações conflitantes em diferentes cópias dos dados, os CRDTs podem garantir que todas as cópias cheguem a um estado consistente sem a necessidade de coordenação centralizada.

Os CRDTs são especialmente úteis em cenários onde a consistência forte entre nós é difícil ou impossível de ser garantida, como em sistemas distribuídos altamente escaláveis ou em ambientes com comunicação intermitente. Eles são aplicados em uma variedade de aplicações, incluindo sistemas de mensagens em tempo real, colaboração em documentos e bancos de dados distribuídos. Ao eliminar a necessidade de resolução de conflitos centralizada, os CRDTs simplificam o desenvolvimento de sistemas distribuídos e reduzem a sobrecarga de comunicação, ao mesmo tempo em que garantem a consistência dos dados em toda a rede.

No contexto do Protocolo Matrix, o uso de CRDTs através do Matrix-CRDT permite a criação de aplicativos e serviços que compartilham dados de forma descentralizada e sincronizada entre diferentes dispositivos e instâncias. Exemplificando, as trocas de mensagens entre os servidores ocorrem sempre que um usuário — seja ele uma pessoa, um *bot* ou um sensor — envia uma mensagem para a sala onde participa da conversação [Shapiro et al. 2011]. Esse envio de mensagem cria um MEG, que é replicado entre todos os servidores participantes da sala. Caso um servidor esteja inacessível no momento do recebimento, ele fará uma sincronização posterior dos eventos da sala e, assim, terá todas as mensagens enviadas para a sala no momento em que estava disponível.

Em termos gerais, o MEG é um grafo direcionado onde cada evento representa uma mensagem que ocorreu em uma sala de chat, os nós representam um evento específico e as arestas representam uma relação entre os eventos. Na Figura 3, é feita uma representação visual de uma troca de mensagens entre dois servidores distintos participando de uma mesma sala de conversação. O estado A representa o último estado consistente da troca de mensagens, ou seja, o estado inicial da troca de mensagens no exemplo. Nota-se que os usuários dos servidores enviaram, em momentos muito próximos suas mensagens para a sala de conversação e essa troca, ocasiona a criação de novos nós no grafo. Esse grafo apresentará momentaneamente inconsistências entre as mensagens dos servidores, mas ainda analisando a figura nota-se que em determinado momento todos os seus nós estarão consistentes, mesmo que em ordem diferente entre os servidores. De uma forma simplificada, mensagens enviadas ao mesmo tempo pelos usuários são organizadas no MEG para sincronização entre os diferentes *homeservers*.

O servidor Synapse basicamente é uma implementação de um *homeserver* Matrix escrito na linguagem Python, com o suporte da biblioteca Twisted [Twisted 2025] para a

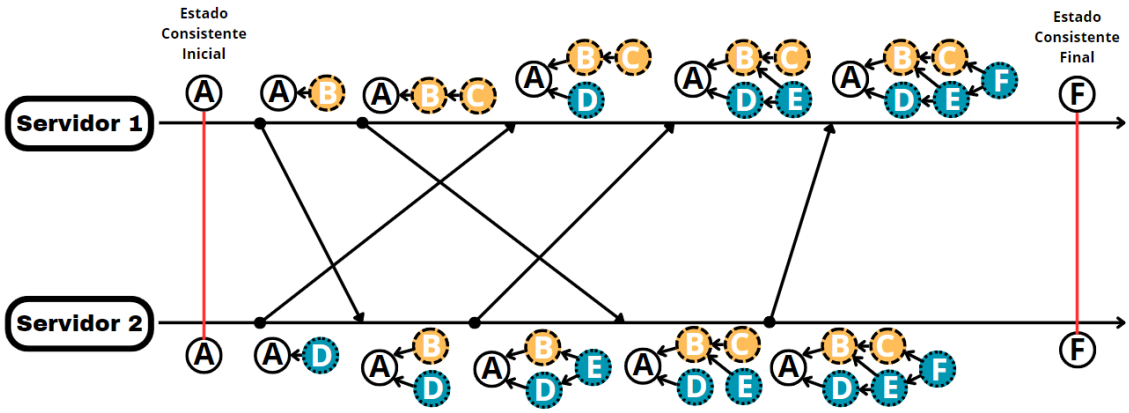


**Figura 3. Exemplo da troca de mensagens do protocolo Matrix.**

camada de acesso à rede. Por ser um protocolo aberto, o responsável pela comunicação pode escolher se utilizará a infraestrutura da fundação Matrix, ou hospedará seu próprio servidor. Existem *homeservers* escritos pela própria Fundação Matrix escritos em Python que respondendo com o codinome Synapse, C++ respondendo com o codinome Construct, Rust sendo chamado de Conduit, Go chamado de Dendrite e C puro chamado de Telodendria [Matrix 2025d]. Entre todos, o único em versão estável é o Synapse, por isso foi o escolhido para o experimento.

## 3. Estudo de Caso

Esta seção apresenta dois estudos de caso com aplicações críticas e onde a capacidade do servidor Synapse de tolerar falhas é fundamental. O primeiro leva em consideração a conversação dos agentes envolvidos (médicos, pacientes, enfermeiros, etc.) em um ambiente de atendimento do sistema público de saúde e o segundo no contexto de Internet das Coisas aplicada ao monitoramento de pacientes hospitalares. Os dois sistemas podem ser complementares, e sua representação visual pode ser vista na Figura 4.

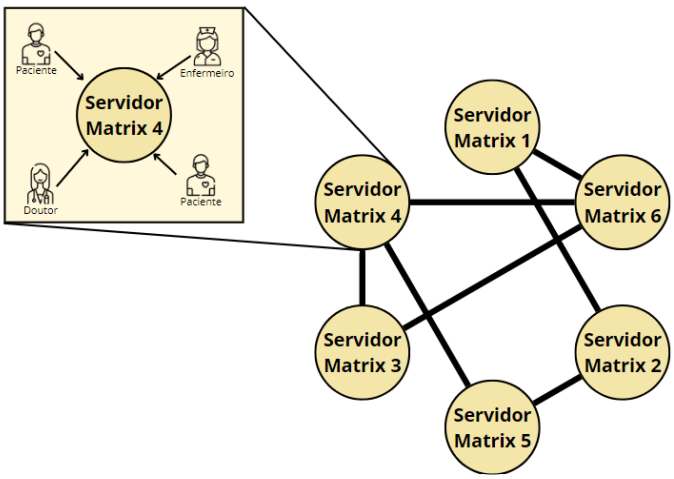


**Figura 4. Estudo de caso**

### 3.1. Sistema de Comunicação para Postos de Saúde

Postos de saúde frequentemente precisam de sistemas de comunicação eficazes para facilitar a coordenação entre equipes médicas, enfermeiros, administradores e outros profissionais de saúde. A alta disponibilidade é essencial para garantir que as comunicações críticas possam ocorrer sem interrupções, mesmo em situações de emergência. Um sistema de comunicação para postos de saúde, baseado no protocolo de comunicação Matrix, oferecerá uma estrutura descentralizada e escalável para a troca de mensagens em tempo real.

O sistema deve estar disponível 24 horas por dia, 7 dias por semana. A escalabilidade também é necessária, pois o sistema deve ser capaz de lidar com um grande volume de mensagens e usuários, especialmente durante picos de atividade. Deve ser intuitivo e fácil de usar, mesmo para usuários não técnicos, como enfermeiros e pessoal administrativo.

O protocolo Matrix pode ser um candidato para desenvolver o sistema descrito, pois possui praticamente todos os requisitos exigidos. Apresenta arquitetura descentralizada, criptografia de ponta a ponta e capacidade de integração. Cada posto de saúde poderia possuir seu próprio *homeserver* a depender do número de usuário daquele posto.

### 3.2. Monitoramento de pacientes hospitalares

Com o avanço da tecnologia IoT, os hospitais estão explorando maneiras inovadoras e eficazes de monitorar e gerenciar seus pacientes. Neste estudo de caso, será abordada a implementação de uma rede IoT baseada no protocolo Matrix para o acompanhamento de pacientes hospitalares, visando proporcionar monitoramento contínuo e em tempo real dos parâmetros vitais dos pacientes.

O sistema proposto consiste em uma rede de dispositivos IoT distribuídos nos quartos dos pacientes, conectados a uma plataforma central. Os dispositivos IoT incluem sensores para monitoramento de sinais vitais, como frequência cardíaca, pressão arterial, temperatura corporal e níveis de oxigênio no sangue. Esses dados são transmitidos em tempo real para a plataforma central, onde são processados e disponibilizados para visualização e análise pelos profissionais de saúde.

Os sensores de monitoramento serão instalados nos quartos dos pacientes. Esses dispositivos serão configurados para se comunicar com a plataforma central responsável por receber, processar e armazenar os dados dos sensores IoT. A plataforma central poderá ser integrada aos sistemas de registro médico eletrônico do hospital para garantir a sincronização e a acessibilidade dos dados dos pacientes. Essa infraestrutura se tornará uma central de alertas e notificações para os profissionais de saúde sobre leituras anormais ou situações de emergência.

Essa implementação de uma rede IoT baseada no protocolo Matrix para o monitoramento de pacientes hospitalares pode trazer uma série de benefícios aos usuários, incluindo monitoramento contínuo e em tempo real, comunicação eficiente entre equipes médicas e maior segurança dos dados dos pacientes. Ao aproveitar as vantagens da IoT e do protocolo Matrix, os hospitais podem melhorar significativamente a qualidade do cuidado ao paciente, reduzir os tempos de resposta em emergências e fornecer um ambiente mais seguro e eficiente para os pacientes e profissionais de saúde.

**Tabela 1. Especificações dos três servidores virtuais utilizados no experimento.**

| | Servidor 1 | Servidor 2 | Servidor 3 |
|---|---|---|---|
| **FQDN** | matrix.td.utfpr.edu.br | synapse.td.utfpr.edu.br | dendrite.td.utfpr.edu.br |
| **Processador** | 1x Xeon E5530 | 1x Xeon E5530 | 1x Xeon E7-4850 |
| **Memória** | 4GB DDR4 | 4GB DDR4 | 4GB DDR4 |
| **Disco** | 100GB SAS | 100GB SAS | 25GB SSD |
| **Rede** | 1GB | 1GB | 1GB |
| **Versão SO** | Debian 12.2 | Debian 12.2 | Debian 12.2 |
| **Synapse** | 1.97 | 1.97 | 1.97 |
| **nGinx** | 1.22.1-9 | 1.22.1-9 | 1.22.1-9 |
| **Hypervisor** | KVM | KVM | Xen |
| **Localização** | UTFPR Toledo | UTFPR Toledo | UTFPR Curitiba |

Considerando os sistemas descritos acima, a seção seguinte avalia o impacto de falhas do tipo parada no servidor Synapse.

## 4. Descrição do Experimento

O experimento consistiu na avaliação da capacidade dos servidores Synapse de se recuperarem após um evento de perda de rede/servidor, simulando uma falha do tipo *crash* (por parada). Espera-se que o servidor seja capaz de recuperar todos os eventos perdidos nas salas em que participa de federação após recuperar o acesso à rede ou mesmo todo o sistema operacional do servidor devido às características providas pelo MEG e o CRDT.

Foram preparadas três máquinas virtuais executando o serviço *Synapse*. Foi criada uma sala de conversação federada entre eles com a criptografia das conversas ativa desde o início. A instalação se deu por meio dos pacotes **.deb** presentes nos repositórios padrão do sistema operacional Debian 12.2. O banco de dados foi mantido como *SQLite* nos quatro primeiros cenários, visto que o desempenho não é um ponto a ser discutido no presente artigo e somente no quinto cenário se configurou uma instância do banco de dados PostgreSQL [PostgreSQL 2025] que é requerida para que o cenário seja válido. Não foi feita federalização com qualquer outro servidor fora da rede de testes. Os FQDN's (*Fully Qualified Host Name*) configurados não expressam as versões dos servidores, foram escolhidos aleatoriamente para registrá-los na rede federada. As especificações completas da rede federada podem ser vistas na Tabela 1.

Em cada servidor, foi criado um usuário para que a comunicação entre todos possa ser visualizada. O diagrama lógico da estrutura criada pode ser visualizado na Figura 1. O processo foi dividido em quatro níveis, cada um apresentando um desafio maior para o servidor Synapse. Todas as simulações de falha foram feitas nos servidores um e dois, ficando o servidor três como experimento controle. Os documentos de instalação e configuração do servidor Synapse [Matrix 2025c] recomendam que ele fique atrás de um software de proxy e não diretamente conectado à Internet por questões de segurança. Essa configuração foi realizada com o servidor nGinx, por sua conhecida robustez e facilidade de utilização. A seguir, os três cenários de avaliação são descritos.

### 4.1. Cenário 1

O cenário inicial consistiu no encerramento repentino do processo do servidor Synapse no servidor 2 com o comando *kill -9 PID*. Esse encerramento repentino durou somente alguns

segundos e foi reiniciado pelo Systemd, como já era esperado, já que a configuração padrão do serviço inclui a *Tag "Restart=always"* nas configurações. Essa *Tag* reinicia o processo assim que ele é encerrado de forma abrupta. A queda do servidor de fato ocorreu, mas como o tempo entre a finalização do servidor e o seu reinício foi insignificante os usuários não perceberam a queda na conexão.

O segundo passo do experimento foi o encerramento repentino do processo do servidor nGinx com o comando *kill -9* PID. No caso do nGinx, o processo não é configurado por padrão com a *Tag "Restart=always"* e assim que o processo foi encerrado permaneceu dessa forma até que foi iniciado novamente após cerca de dois minutos. A captura das conversações dos três clientes pode ser vista na Figura 5. Nota-se que mesmo fora de ordem, as mensagens recebidas para o usuário 2 foram (1, 4, 5, 2, 3) e todas as conversas se sincronizaram posteriormente ao retorno do servidor. Essa ordenação se explica pelo protocolo Matrix ser um SEC, onde se garante que todas as mensagens serão entregues, mas não se garante a ordenação das mesmas. Uma solução para contornar o problema é garantir que, por padrão, o servidor Nginx esteja configurado com a *Tag "Restart=always"* habilitada. Com essa alteração implementada, o comando para encerrar o processo do Nginx deixou o servidor fora de alcance para os clientes por apenas alguns segundos.


**Figura 5. Conversação entre os usuários durante queda do servidor Synapse.**

## 4.2. Cenário 2

No segundo cenário, foi simulada uma parada total do sistema operacional do servidor 2, desligando-o manualmente com um comando de *"Power Off"* através do *hypervisor*. Na Figura 6 pode-se ver que foi tomado o cuidado de desligar o servidor durante o ato de escrita da mensagem, ou seja, o cliente iniciou a composição do texto enquanto o servidor estava ativo, mas o envio da mensagem só ocorreu após a queda. Obviamente, não obteve sucesso no envio da mensagem durante o período de indisponibilidade.

Após alguns minutos, o servidor foi ligado novamente, sendo necessária uma verificação do estado dos discos já que o processo "*Power Off*" forçado não realiza um desligamento correto. Essa sequência de verificação levou cerca de quinze minutos para ser concluída, deixando o cliente do servidor 2 incomunicável durante todo esse tempo. Após a sequência de *boot* o servidor Synapse voltou à ativa. Todas as mensagens foram sincronizadas com a comunidade federada, já que em seu retorno o servidor envia uma requisição aos demais sobre os eventos perdidos por ele. Novamente, as mensagens foram entregues em sua totalidade, mas fora de ordem.

Entre as soluções que poderiam resolver o problema do cenário 2, destaca-se a execução em uma plataforma de contêiner, como Docker [Docker 2025] ou OpenShift [RedHat 2025]. No entanto, a implementação de conteinerização está além do escopo deste trabalho.

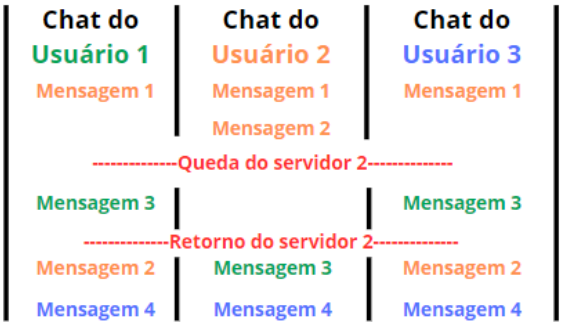


**Figura 6. Nova conversação entre os usuários durante queda do Synapse.**

### 4.3. Cenário 3

O terceiro cenário envolveu a perda drástica do servidor 1. Em um ambiente de produção, recomenda-se que o banco de dados seja instalado em um servidor separado. Ao invés disso, para fins de simplicidade, foi feita uma cópia do banco *SQLite* e dos arquivos de configuração do servidor.

Na sequência, o servidor 1 foi formatado e uma versão limpa do sistema operacional *Debian* 12.2 instalada com todos os pacotes padrão, de forma similar ao que já havia sido feito no início do experimento. Novamente, os passos recomendados pelo manual de instalação do servidor Synapse foram seguidos e a cópia do banco de dados foi restaurada. Dessa forma, simulou-se a perda do servidor, mas não do seu banco de dados.

O servidor Synapse se mostrou resiliente e, como nos três cenários anteriores, o usuário restaurou a conexão e as mensagens foram entregues, como pode ser visto na Figura 7. Alternativas para aumentar a disponibilidade podem incluir a separação do banco de dados em *cluster* e a configuração de mais de um servidor Synapse no modelo Ativo/Passivo. Porém, tal ação não é documentada como padrão da plataforma.

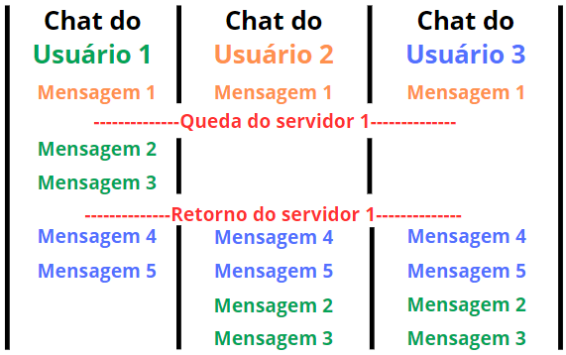


**Figura 7. Conversação durante queda total do servidor Synapse.**

### 4.4. Cenário 4

O quarto cenário envolve uma falha catastrófica do servidor 2, com perda das configurações e do banco de dados. O maior problema dessa falha é que, como o banco de dados foi perdido, todos os usuários que faziam parte desse servidor também o foram. Dessa forma, somente se todos os usuários forem recriados manualmente é que conseguirão logar no servidor.

A chave criptográfica, conhecida como *Security Key*, responsável por validar o dispositivo que o usuário está usando, fica em posse do usuário e é necessária sempre que um novo dispositivo acessa a conta. Nesse caso extremo, mesmo que o usuário utilize sua *Security Key* para validar seu dispositivo, não conseguirá ler suas mensagens, já que, do ponto de vista do servidor, esse é um novo usuário e seu par de chaves não será descriptografado.

Dessa forma, por ser basicamente uma nova instalação, todas as mensagens trocadas anteriormente serão perdidas pelos usuários nesse cenário. Assim, destaca-se a importância da proteção do banco de dados para que o servidor se recupere após falhas, já que toda a infraestrutura de comunicação dependerá de um banco de dados robusto e altamente disponível. Sistemas de banco de dados como PostgreSQL, possuem bom desempenho e são, inclusive, recomendados pela documentação do servidor Synapse para utilização em produção. A configuração de replicação e rotinas de backup eficientes dificilmente levariam os usuários a perder seus históricos de conversação, sendo essa a recomendação para o cenário 4.

### 4.5. Cenário 5

Este cenário avaliou a adoção de *Workers*. O conceito de *workers*, descrito na Seção 2, consiste em criar mais um processo Python para distribuir a carga de trabalho, sendo esses processos executados na mesma máquina ou em *hosts* separados. Para o experimento, optou-se por rodar todos os processos em uma mesma máquina, e uma visualização da forma como foi implementada pode ser vista na Figura 8.

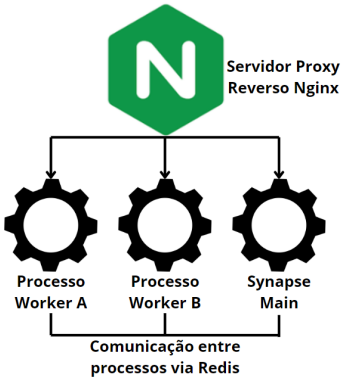


**Figura 8. Diagrama visual da implementação realizada.**

De forma geral, os *workers* dividem o processo *Main* em pedaços menores que são responsáveis por distribuir a carga de trabalho. O servidor nGinx recebe a requisição da Internet e usando escalonamento circular (*round robin*) as distribui entre os processos A e B. Ficando o processo *Synapse Main* responsável pela coordenação dos *workers*. A ideia

desse cenário consiste em identificar o que pode acontecer com os clientes caso algum dos processos seja interrompido.

Inicialmente simulou-se a falha por parada no Processo Worker A com o comando *kill -9 PID*. Notou-se que as mensagens não foram perdidas durante esse processo. Porém, como o *proxy* nGinx não tem conhecimento do processo encerrado, continua transmitindo as solicitações. Para contornar o problema, o servidor nGinx foi configurado com checagens de saúde dos Workers A e B, solucionando a questão. Os tempos configurados da checagem de saúde foram *max_fails*=3 *fail_timeout*=30s. Graças à arquitetura MEG do protocolo Matrix as mensagens que não foram entregues foram sincronizadas após alguns momentos.

A última avaliação consistiu em finalizar o processo Synapse_Main, também com o comando *kill -9 PID*. A finalização desse processo acarretou a queda do servidor para os clientes de forma quase imediata, da mesma forma que foi observada no cenário 1. Ou seja, o uso de workers pode aumentar o desempenho e a distribuição de tarefas, mas a falha do processo *Main* ainda ocasiona a indisponibilidade do servidor.

## 5. Considerações Finais

Este trabalho investigou o comportamento do servidor Synapse perante falhas do tipo parada. Os resultados apresentados em cinco cenários sugerem que a falha de um único servidor não afeta a conversação dos demais usuários conectados em outros servidores. Ainda, caso o servidor se recupere após uma falha, foi verificado que o protocolo Matrix permite ao servidor recuperar as mensagens, recebendo-as de outros usuários da federação. Verificou-se ainda a importância de manter o banco de dados disponível para que a comunicação continue apesar da falha do servidor. Para os estudos de caso apresentados, onde há a descrição de aplicações críticas, é essencial evitar a indisponibilidade do sistema. Dessa forma, como trabalhos futuros, uma alternativa é aumentar a disponibilidade do servidor por meio de abordagens como a replicação passiva ou ativa.

## Referências


Al-Masri, E., Kalyanam, K. R., Batts, J., Kim, J., Singh, S., Vo, T., and Yan, C. (2020). Investigating messaging protocols for the internet of things (iot). *IEEE Access*, 8:94880–94911.

Albrecht, M. R., Celi, S., Dowling, B., and Jones, D. (2023). Practically-exploitable cryptographic vulnerabilities in matrix. Cryptology ePrint Archive, Paper 2023/485. `https://eprint.iacr.org/2023/485`.

Ausat, A. and Almaududi, M. (2023). The role of social media in shaping public opinion and its influence on economic decisions. *Technology and Society Perspectives (TACIT)*, 1:35–44.

Avizienis, A., Laprie, J.-C., Randell, B., and Landwehr, C. (2004). Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33.

Cao, Y., Wei, J., Huang, X., Chen, X., and Xiang, Y. (2024). Deniable identity-based matchmaking encryption for anonymous messaging. *IEEE Transactions on Dependable and Secure Computing*.

Chatwoot (2025). Chatwoot customer engagemento suite. https://www.chatwoot.com/. Accessado: (20/03/2025).

Discourse (2019). Synchronous messaging at mozilla: The decision. https://discourse.mozilla.org/t/synchronous-messaging-at-mozilla-the-decision/50620. Accessado: (22/03/2025).

Docker (2025). Docker: Accelerated container application development. https://www.docker.com/. Accessado: (20/03/2025).

Domenech, M. C., Abed Grégio, A. R., and Erpen de Bona, L. C. (2022). On metadata privacy in instant messaging. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–7.

El-Dardiry, Y. (2022). Matrix crdt. https://github.com/YousefED/Matrix-CRDT. Accessado: (22/03/2025).

Element (2025). Element secure communication platform. https://element.io/. Accessado: (20/03/2025).

Heise (2019). Open source: Bundeswehr baut eigene verschlüsselte messenger-app. https://www.heise.de/news/Open-Source-Bundeswehr-baut-eigene-verschluesselte\n-Messenger-App-4623404.html. Accessado: (22/03/2025).

Ihle, C., Deifuß, F., Schubotz, M., and Gipp, B. (2022). Towards portable identities in the matrix protocol. In *2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 88–89.

Jacob, F., Beer, C., Henze, N., and Hartenstein, H. (2021). Analysis of the matrix event graph replicated data type. *IEEE Access*, 9:28317–28333.

Jacob, F., Grashöfer, J., and Hartenstein, H. (2019). A glimpse of the matrix: Scalability issues of a new message-oriented data synchronization middleware. *Middleware Demos and Posters 2019 - Proceedings of the 2019 20th International Middleware Conference Demos and Posters, Part of Middleware 2019*, pages 5–6.

Karhu, J. (2023). German health professionals will communicate with each other through the open source matrix protocol. https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/luxembourg-launches-open-source-chat-officials-and-citizens. Accessado: (22/03/2025).

Kleppmann, M. (2022). Making crdts byzantine fault tolerant. In *Proceedings of the 9th Workshop on Principles and Practice of Consistency for Distributed Data*, PaPoC '22, page 8–15, New York, NY, USA. Association for Computing Machinery.

Matrix (2025a). Elements of matrix. https://matrix.org/docs/matrix-concepts/elements-of-matrix/#homeserver. Accessado: (20/02/2025).

Matrix (2025b). Frequently asked questions. https://matrix.org/docs/older/faq/. Accessado: (05/04/2025).

Matrix (2025c). Installation instructions. https://matrix-org.github.io/synapse/latest/setup/installation.html. Accessado: (20/02/2025).

Matrix (2025d). Servers of matrix. https://matrix.org/ecosystem/servers/. Accessado: (20/02/2025).

Nelson, B., Pagnin, E., and Askarov, A. (2024). Metadata privacy beyond tunneling for instant messaging. In *Proceedings - 9th IEEE European Symposium on Security and Privacy, Euro S and P 2024*, pages 697–723. Institute of Electrical and Electronics Engineers Inc.

Pastório, A., Rodrigues, L., and de Camargo, E. (2020). Uma revisão sistemática da literatura sobre tolerância a falhas em internet das coisas. In *Anais Estendidos do X Simpósio Brasileiro de Engenharia de Sistemas Computacionais*, pages 57–64, Porto Alegre, RS, Brasil. SBC.

PostgreSQL (2025). Postgresql relational database. https://www.postgresql.org/. Accessado: (26/03/2025).

Preguiça, N., Baquero, C., and Shapiro, M. (2019). *Conflict-Free Replicated Data Types CRDTs*, pages 491–500. Springer International Publishing, Cham.

Pätsch, S. (2021). German health professionals will communicate with each other through the open source matrix protocol. https://joinup.ec.europa.eu/collection/open-source-observatory-osor/news/german-health-system-adopts-matrix. Accessado: (22/03/2025).

RedHat (2025). Red hat openshift. https://www.redhat.com/pt-br/technologies/cloud-computing/openshift. Accessado: (20/03/2025).

Schipper, G. C., Seelt, R., and Le-Khac, N.-A. (2021). Forensic analysis of matrix protocol and riot.im application. *Forensic Science International: Digital Investigation*, 36:301118. DFRWS 2021 EU - Selected Papers and Extended Abstracts of the Eighth Annual DFRWS Europe Conference.

Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M. (2011). Conflict-free replicated data types. In *Stabilization, Safety, and Security of Distributed Systems*, pages 386–400, Berlin, Heidelberg. Springer Berlin Heidelberg.

Sparber, J. (2020). Blockchain-based end-to-end encryption for matrix instant messaging.

Sule, M.-J., Zennaro, M., and Thomas, G. (2021). Cybersecurity through the lens of digital identity and data protection: Issues and trends. *Technology in Society*, 67:101734.

Tchap (2025). Tchap, instant messaging for the public sector. https://tchap.beta.gouv.fr/?mtm_campaign=TchapWebConnectLogo. Accessado: (20/03/2025).

Twisted (2025). Twisted: An event-driven networking engine. https://twisted.org/. Accessado: (27/02/2025).

Zulip (2025). Zulip chat and collaborative open source platform. https://zulip.com/. Accessado: (20/03/2025).