UNIVERSIDADE FEDERAL DA FRONTEIRA SUL
CAMPUS DE CHAPECÓ
CURSO DE CIÊNCIA DA COMPUTAÇÃO

NICOLAS KOLLING RIBAS

FEDERAÇÃO DE BROKERS DO PROTOCOLO MQTT

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO

CHAPECÓ
2022

NICOLAS KOLLING RIBAS

FEDERAÇÃO DE BROKERS DO PROTOCOLO MQTT

IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.
Orientador: Prof. Dr. Marco Aurélio Spohn

CHAPECÓ
2022

Ribas, Nicolas Kolling

Federação de brokers do protocolo MQTT: Implementação e análise de desempenho / Nicolas Kolling Ribas. – 2022.
37 f.: il.

Orientador: Prof. Dr. Marco Aurélio Spohn.
Trabalho de conclusão de curso (graduação) – Universidade Federal da Fronteira Sul, curso de Ciência da Computação, Chapecó, SC, 2022.

1. Federação. 2. Escalabilidade. 3. *MQTT*. 4. *Broker*. I. Spohn, Prof. Dr. Marco Aurélio, orientador. II. Universidade Federal da Fronteira Sul. III. Título.

E-mail: nicolaskribas@gmail.com

NICOLAS KOLLING RIBAS

FEDERAÇÃO DE BROKERS DO PROTOCOLO MQTT
IMPLEMENTAÇÃO E ANÁLISE DE DESEMPENHO

Trabalho de conclusão de curso apresentado como requisito para obtenção do grau de Bacharel em Ciência da Computação da Universidade Federal da Fronteira Sul.

Orientador: Prof. Dr. Marco Aurélio Spohn

Este trabalho de conclusão de curso foi defendido e aprovado pela banca avaliadora em: 30/03/2022.

BANCA AVALIADORA

Prof. Dr. Marco Aurélio Spohn – UFFS

Prof. Me. Adriano Sanick Padilha – UFFS

Prof. Dr. Braulio Adriano de Mello – UFFS

RESUMO

A implantação de bilhões de dispositivos inteligentes capazes de sentir e interagir com seu ambiente, através da aquisição e envio de dados pela rede é conhecido como a Internet das Coisas. Dentre os protocolos que possibilitam a comunicação destes dispositivos, o mais difundido é o Message Queuing Telemetry Transport (MQTT). Em sua implementação padrão o MQTT faz uso de apenas um servidor central (i.e. *broker*) para a filtragem e repasse de mensagens. O *broker* se apresenta, nesse contexto, como um ponto único de falha e um possível gargalo no desempenho do sistema. Surge assim a necessidade de abordagens escaláveis do MQTT, como a *clusterização* e a federação de *brokers*. O presente trabalho possui como objetivo analisar o desempenho da federação de *brokers* MQTT. Para tal, apresentamos uma nova implementação de um federador: aplicação que fornece as capacidades de federação aos *brokers*. Utilizando do federador, então, montamos um estudo de caso, onde comparamos o desempenho das abordagens tradicional (de único *broker*) e de federação, tendo como métrica a latência de publicações.

Palavras-chave: Federação. Escalabilidade. *MQTT*. *Broker*.

ABSTRACT

The deployment of billions of intelligent devices capable of sensing and interacting with their environment by acquiring and sending data over the network is known as the Internet of Things. Among the protocols that enable the communication of these devices, the most widespread is the Message Queuing Telemetry Transport (MQTT). MQTT uses only one central server (i.e., broker) for filtering and forwarding messages in its standard implementation. The broker presents itself, in this context, as a single point of failure and a possible bottleneck in system performance. Thus arises the need for scalable MQTT approaches, such as clusterization and federation of brokers. The present work aims to analyze the federation's performance of MQTT brokers. To this end, we present a new implementation of a federator: an application that provides federation capabilities to brokers. Using the federator, we set up a case study comparing the performance of the traditional (single broker) and federation approaches, using the publication latency as a metric.

Keywords: Federation. Scalability. MQTT. Broker.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

Possibilitada pelo avanço no desenvolvimento de sensores, tecnologias de comunicação e protocolos de internet, a Internet das Coisas (IoT) é o conceito que descreve uma conexão ubíqua à internet, tornando objetos comuns do cotidiano em dispositivos interconectados. A ideia central desse conceito é a da implantação de bilhões de dispositivos inteligentes capazes de sentir e interagir com seu ambiente, através da aquisição e envio de dados pela rede. Um exemplo de aplicação da IoT está nas casas inteligentes, nas quais há a possibilidade de se monitorar e controlar remotamente os objetos domésticos, que com sensores identificam seu próprio estado (como uma lâmpada, acesa ou apagada), e com atuadores o modificam. Para além disso, a aplicação da IoT é vasta, sendo utilizada nas mais variadas indústrias como varejo, manufatura, saúde, seguros, eletrodomésticos, equipamentos pesados, companhias aéreas e logística (4).

A comunicação entre os diversos dispositivos da IoT é feita, principalmente, através de protocolos de comunicação Máquina-a-Máquina (M2M), dentre eles se destaca o *Message Queuing Telemetry Transport* (MQTT) como sendo o mais difundido (6). O MQTT utiliza o padrão Publicador-Assinante (P/S): dispositivos interessados em enviar alguma informação (i.e. publicadores) fazem a publicação da mesma em um servidor central, chamado de *broker*, que a a envia para os dispositivos interessados em receber a informação (i.e. assinantes). Neste contexto de implementação padrão do MQTT onde é feito o uso de apenas um *broker*, ele se apresenta como um ponto único de falha e um possível gargalo no desempenho do sistema.

A fim de obter-se uma maior escalabilidade no MQTT, pode-se recorrer a abordagens em que há a utilização de múltiplos *brokers* por sistema, possibilitando uma maior tolerância à falha e aumento de desempenho. Na *clusterização*, um elemento balanceador de carga age como ponto de entrada para os clientes, as solicitações são feitas diretamente para o balanceador de carga que então as distribui entre os *brokers* do *cluster*. Outra abordagem, alvo deste trabalho, se baseia na criação de uma federação de *brokers*, na qual os diferentes *brokers* mantêm conexão entre si e se auto-organizam para entregar as publicações feitas em um *broker* aos assinantes correspondentes, não importando a qual outro *broker* eles estão associados.

A primeira solução para a federação foi apresentada por Spohn (9) e tem como ideia central a de *brokers* com assinantes locais criarem e manterem uma estrutura de malha que os interconecta e possibilita o roteamento de publicações entre eles. Para a implementação de tal solução eram necessárias modificações no funcionamento interno dos *brokers*, assim Spohn (8) propôs uma nova variante à sua solução original, introduzindo o conceito de federador: uma aplicação, que associada ao *broker*, dá a ele os mecanismos necessários para a realização da federação. Deste modo o federador fica responsável pela criação e manutenção das malhas como também do roteamento de mensagens para outros *brokers*.

2 OBJETIVOS

2.1 OBJETIVO GERAL

Avaliar o desempenho de um protocolo de federação de *brokers* MQTT.

2.2 OBJETIVOS ESPECÍFICOS

- Realizar a implementação de um federador de *brokers* MQTT;
- Avaliar, com a utilização de métricas, o desempenho da implementação da federação de *brokers* em relação a abordagem tradicional de único *broker*.

2.3 JUSTIFICATIVA

Atualmente, bilhões de dispositivos fazem parte da IoT e as perspectivas são de crescimento. Estima-se que até 2030 o números de dispositivos IoT conectados no mundo dobrará, alcançando um total de 28 bilhões (12). Para lidar com o grande volume de dados produzidos e transmitidos por estes dispositivos, soluções de escalabilidade devem ser desenvolvidas.

De acordo com Mishra; Kertesz (6), conforme o mundo se move em direção à computação distribuída e computação de borda, um maior número de pesquisas em implementações escaláveis do protocolo MQTT se faz necessário. A solução de federação de *brokers* MQTT é promissora tanto em questão de escalabilidade e de confiabilidade, podendo superar as já estabelecidas soluções baseadas em *clusterização*. Mas ainda há espaço para análises aprofundadas da sua performance. Deste modo se faz oportuna e relevante a realização de uma implementação e analise de desempenho de uma federação de *broker* a fim de demonstrar suas possibilidades quanto a escalabilidade.

3 REVISÃO BIBLIOGRÁFICA

3.1 MQTT

Message Queuing Telemetry Transport (MQTT) é um protocolo de transporte de mensagens criado por Andy Stanford-Clark e Arlen Nipper em 1999, tendo sua versão mais recente (i.e. MQTT v5) padronizada em 2019 pela *Organization for the Advancement of Structured Information Standards* (10). O design do protocolo tem como foco a minimização do uso de banda de rede e de recursos dos dispositivos que o implementam garantindo entrega confiável (6).

O protocolo MQTT adota o modelo de Publicador-Assinante (P/S) o que proporciona flexibilidade e facilidade de implementação (3). Em um sistema de mensagens do tipo P/S as mensagens são filtradas por tópicos de interesse, ou seja, um dispositivo que deseja enviar uma mensagem (ie. publicador) não a envia diretamente para um destino especifico mas faz a publicação da mesma em um tópico, sem o conhecimento de quem são seus assinantes. Assinantes, nesse contexto, são todos os dispositivos que tem interesse no tópico e realizaram sua assinatura para receberem as mensagens.

Ao fazer assinaturas, clientes MQTT podem fazer uso de alguns caracteres especiais, chamados de curingas, permitindo receber publicações de múltiplos tópicos com apenas uma assinatura. Tópicos MQTT podem conter múltiplos níveis, cada nível é separada por uma barra (’/’), criando uma estrutura hierárquica de tópicos, nessa estrutura entram em ação os caracteres curingas, que podem substituir os níveis de um tópico. Os dois curingas disponíveis são o curinga multi-nivel (’#’) e o curinga nível único (’+’). Ao utilizar o curinga multi-nível para assinar o tópico "casa/cozinha/#"o cliente receberia publicações de "casa/cozinha", "casa/cozinha/temp"e "casa/cozinha/humid, por exemplo. Já ao utilizar o curinga de nível único para assinar o tópico "casa/+/temp"o cliente receberia publicações de "casa/sala/temp"e "casa/quarto/temp", por exemplo.

Na arquitetura do MQTT, como mostrado na Figura 1, a filtragem e distribuição das mensagens é feita de forma centralizada por um servidor conhecido como *broker*. Os publicadores e assinantes (denominados clientes MQTT) tem conexão direta com o *broker* e fazem suas publicações e assinaturas através dele.

Para a entrega de mensagens o MQTT oferece três níveis de qualidade de serviço (QoS). No primeiro nível (QoS 0), também conhecido como "até uma", não há garantia de entrega da mensagem. Na QoS 1 é empregado a retransmissão, garantindo a entrega das mensagens mas com a possibilidade de duplicatas, por isso é chamado de "no mínimo uma". QoS 2 emprega tanto a retransmissão quanto o controle de mensagens duplicadas garantindo a entrega de "exatamente uma"mensagem.

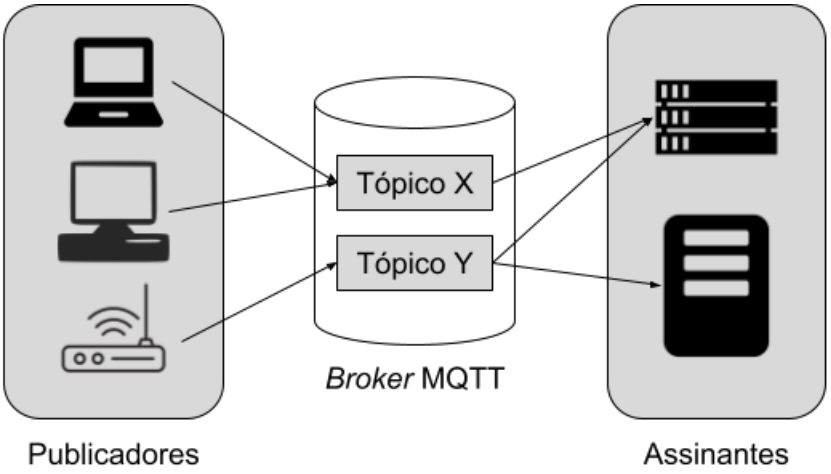

Figura 1 – Arquitetura MQTT

3.2 ESCALABILIDADE DO *MQTT*

Em sua implantação básica o protocolo MQTT descreve o uso de apenas um único *broker* por sistema. Neste cenário o *broker* se apresenta como um ponto único de falha e potencial gargalo no desempenho do sistema. Para abordar a necessidade de performance e disponibilidade do protocolo existem algumas estratégias de escalabilidade, que podem ser divididas em escalabilidade vertical e horizontal.

3.2.1 Escalabilidade Vertical

Na escalabilidade vertical busca-se aumentar a quantidade de recursos no ambiente local, como poder de processamento e/ou capacidade de armazenamento. Também busca-se o uso mais inteligente e eficiente dos recursos disponíveis.

Um exemplo seria o muMQ (7), que é uma implementação de *broker* MQTT que consegue utilizar eficientemente os recursos de CPUs *multi-core*. Ao fazer uso de uma pilha TCP a nível de usuário, é capaz de mapear cada *thread* do *broker* para uma *thread* TCP possibilitando paralelismo. muMQ também adota uma arquitetura orientada a eventos, possibilitando que cada *thread* do *broker* seja capaz de lidar com diversas conexões TCP concorrentemente.

Outra abordagem de aumento de performance é a *clusterização*. Na *clusterização* vertical, múltiplas instâncias do *broker* na mesma máquina trabalham em conjunto para atender os clientes. A demanda é distribuída para os *brokers* do *cluster* por um balanceador de carga que age como um ponto único de entrada para todos os clientes, como mostrado na Figura 2.

3.2.2 Escalabilidade Horizontal

Técnicas de escalabilidade vertical trazem como beneficio um aumento de desempenho, e no caso dos *cluster* de *brokers* há uma certa *redundância* no sistema. Mas para atingirmos

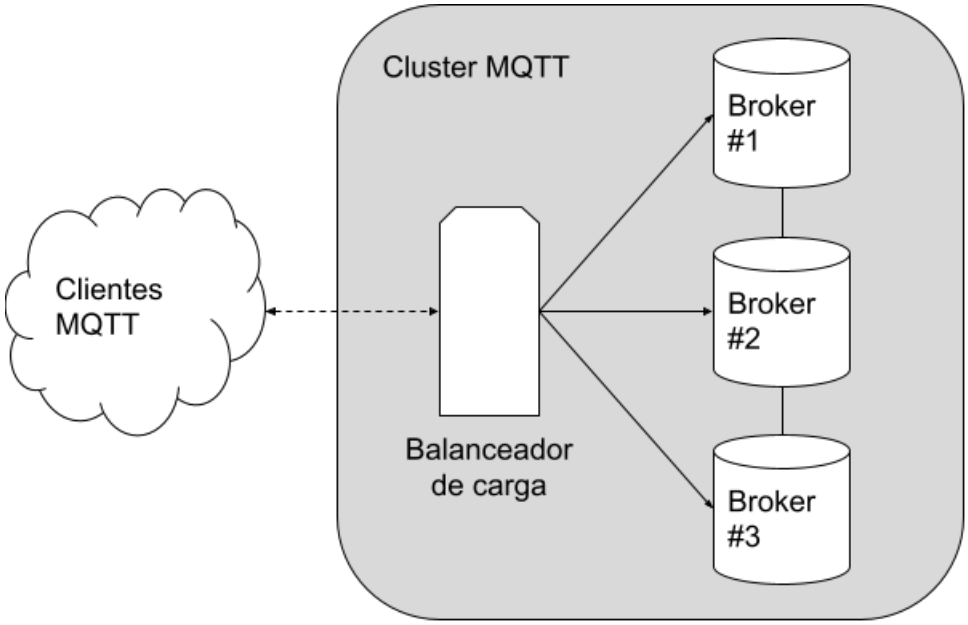

Figura 2 – *Cluster* (exemplo)

uma maior tolerância à falha e superarmos os limites de recurso que apenas uma maquina pode nos fornecer, devemos partir para abordagens de escalabilidade horizontal, podendo ser através de *clusterização* horizontal ou através da federação de *brokers*.

Como na *clusterização* vertical, a *clusterização* horizontal é um conjunto de diversas instâncias do *broker*, mas distribuídos horizontalmente, ou seja, em maquinas diferentes. Isso possibilita uma maior tolerância à falha em comparação a *clusterização* vertical pois não há a dependência da integridade de uma única máquina.

Já em uma federação, um conjunto de *brokers* instanciados em diferentes máquinas inter-operam de forma conjunta e autônoma. Não havendo a necessidade de uma unidade de balanceamento de carga, a federação se auto organiza de forma a garantir que os clientes possam fazer solicitações para qualquer um dos *brokers* da federação.

3.3 FEDERAÇÃO DE *BROKERS*

Spohn (9) propôs a primeira solução para a federação de *brokers*. Em sua solução, *brokers* que possuem inscrições em um tópico em comum se auto-organizam criando uma estrutura de malha que os inter-conecta. Uma malha possui um *broker* central (i.e., núcleo) que é eleito com a função de fazer a construção e manutenção da mesma. Assim que um *broker* recebe a inscrição de um novo tópico ele passa a se anunciar como sendo o núcleo de uma nova malha. Esse anúncio de núcleo é propagado por toda a federação, possibilitando que todos os *brokers* participantes tenham conhecimento de como alcançar o núcleo de qualquer malha. Para aderir a malha, *brokers* com inscritos locais enviam um anúncio de membro de volta para o núcleo pelos caminhos mais curtos até ele (o número de caminhos diferentes pode ser ajustado de acordo com a redundância necessária). *Brokers* que fazem parte do caminho

também aderem a malha, sendo assim, participam de uma malha todos os *brokers* que tem inscritos locais ou aqueles que os ligam até o núcleo. Novas publicações feitas em qualquer um dos *brokers* da federação serão roteados até o núcleo da malha correspondente. Ao chegar em qualquer membro da malha, a publicação então é propagada por toda a malha. Possibilitando assim que as publicações cheguem em todos os subscritos independentemente de qual *broker* eles estão vinculados.

A Figura 3 demonstra o processo de construção de uma malha em uma topologia composta por 6 *brokers* e com valor de redundância igual a 2. O *broker* 1 ao receber uma inscrição local desencadeia o processo de criação da malha para o respectivo tópico, enviando um anúncio de núcleo para os *brokers* vizinhos que irão repassar o anúncio até atingir todos os *brokers* da federação. Assim todos os *brokers* tomam conhecimento do núcleo da malha e como alcança-lo. A malha no momento é composta apenas pelo *broker* 1, mas ao receber uma inscrição local, o *broker* 5 então envia um anúncio de membro para os seus pais (*brokers* 3 e 4 dado a redundância igual a 2). Os *Brokers* 3 e 4 ao receberem o anúncio de membro de 5 passam a fazer parte da malha, também repassando o anúncio de membro para seus pais, que nesse caso é próprio núcleo da malha.

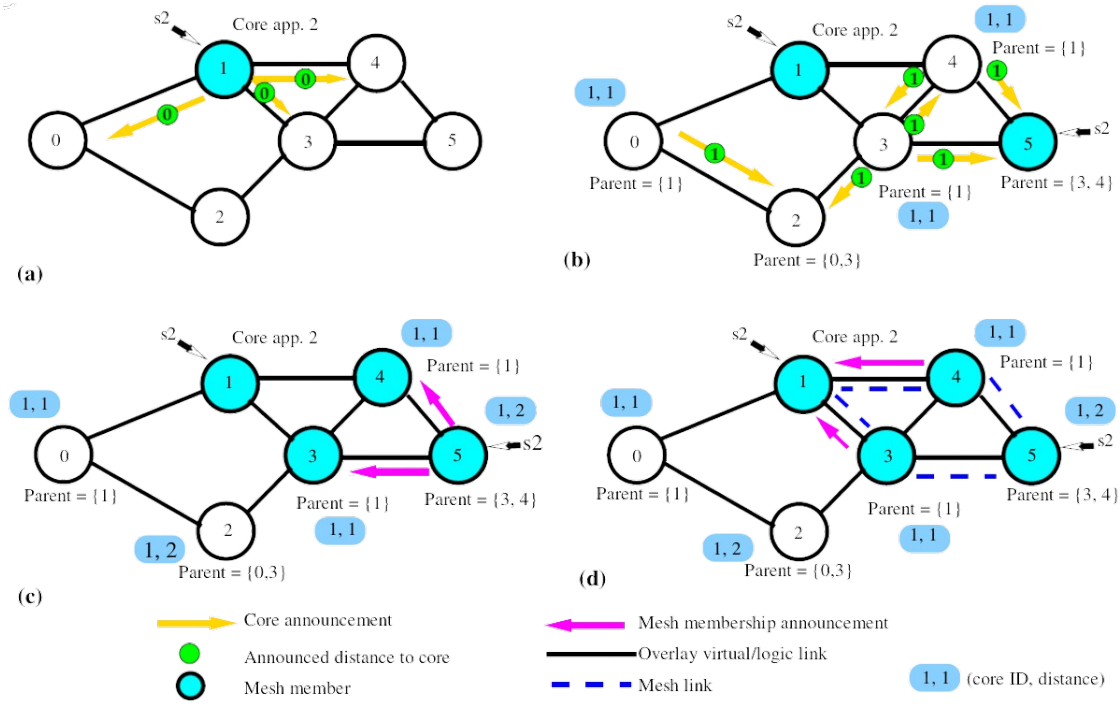

Figura 3 – Processo da construção da malha (Fonte: (9))

A Figura 4 demonstra o roteamento de duas publicações na malha recém construída. A primeira publicação é feita no *broker* 4, como ele pertence a malha correspondente ao tópico da publicação então ele faz o roteamento da mensagem para todos os seus vizinhos também participantes da malha, sendo eles o *broker* 5 (de quem recebeu anúncio de membro) e o *broker* 1 (o qual é um de seus pais). Os *brokers* vizinhos também fazem o mesmo, enviando a seus próprios vizinhos participantes da malha, assim a publicação acaba por atingir todos os *brokers*

da malha. A segunda publicação é recebida pelo *broker* 0, como ele não faz parte da malha ele identifica o núcleo responsável por ela (i.e. *broker* 1) e encaminha a publicação para ele. Ao chegar no núcleo a publicação é encaminhada para todos os vizinhos participantes da malha e assim sucessivamente até a publicação ter atingido todos os membros da *malha*.

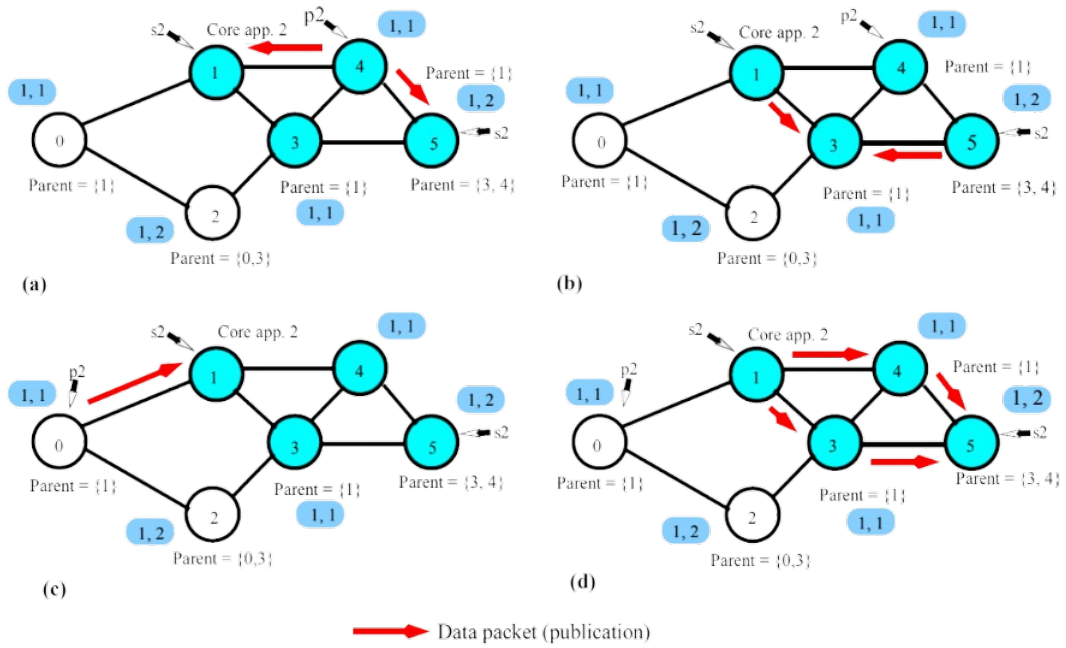

Figura 4 – Processo de roteamento de publicações (Fonte: (9))

3.4 TRABALHOS RELACIONADOS

Em sua solução original, mudanças no funcionamento dos *broker* eram necessárias para que a federação pudesse ser realizada. Com isso, Spohn (8) propôs uma nova variante do protocolo capaz de fornecer os mecanismos de federação ao *broker* sem que houvesse a necessidade de alterações nos mesmos. Para isso foi introduzido o conceito de **federador**: uma aplicação associada ao *broker* que irá realizar tanto a criação e gerenciamento das malhas como o roteamento de mensagens de controle e publicações. O federador é composto por dois sub-processos:

- Pub_Fed: Mantém conexão direta com todos os *brokers* vizinhos sendo responsável pelo envio de mensagens de controle e pelo roteamento de mensagens regulares.
- Sub_Fed: Lida com mensagens de controle recebidas dos vizinhos e também com as mensagens regulares.

A Figura 6 demonstra a implementação do proposto federador. Sub_Fed recebe os anúncios de núcleo e de membros através das publicações de um *broker* federado vizinho em tópicos de controle definidos previamente (*CORE_ANN* e *MESH_MEMB_ANN* respectivamente). Sub_Fed

também está inscrito em todos os tópicos de mensagens regulares, assim, havendo novas publicações ele é capaz de repassa-las para Pub_Fed que por sua vez ira fazer o encaminhamento dessa publicações para os vizinhos enviando a publicação no tópico *DATA* dos mesmos. Apesar de não haver a necessidade de mudanças no *broker*, a solução necessita que os clientes do *broker* atendam a um requisito: eles devem obrigatoriamente informar através de um tópico de controle (*NEW_REGULAR_TOPIC*) toda nova inscrição ou primeira publicação em um tópico regular. Isso é necessário para que o federador fique ciente de quais tópicos ele deve tratar. Caso o cliente do *broker* não atenda a esse requisito não há nenhuma garantia de que suas publicações chegarão a inscritos associados a outros *brokers* da federação.

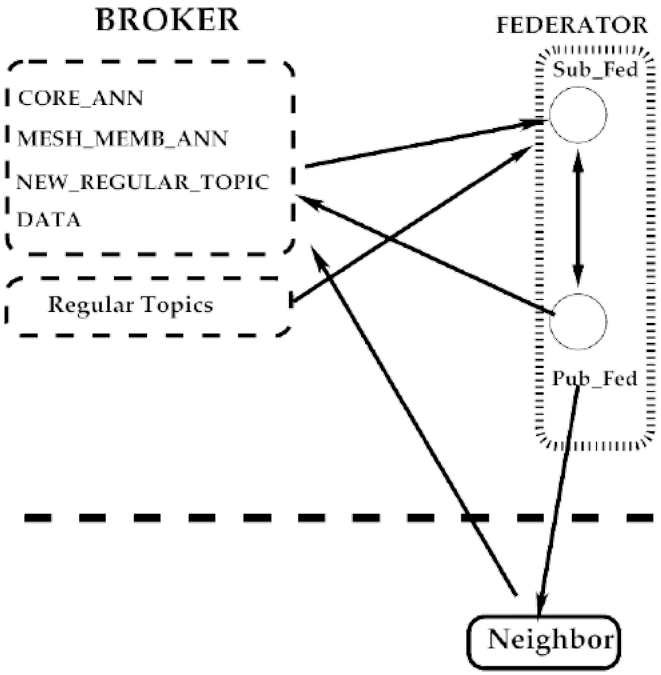

Figura 5 – Federador MQTT (Fonte: Spohn (8))

4 DESENVOLVIMENTO

Para fornecer as capacidades de federação aos *brokers* realizamos a implementação de um federador com base no protocolo proposto por Spohn (8), que se utiliza apenas dos mecanismos nativos de publicação e assinatura do MQTT. Mudanças foram feitas ao protocolo, efetivamente dando origem à uma nova versão. Os detalhes de implementação bem como as mudanças feitas ao protocolo original são descritas a seguir.

4.1 FERRAMENTAS

4.1.1 Eclipse Paho

Paho (1) é um projeto da Eclipse Fundation que visa providenciar implementações de código aberto de clientes do protocolo MQTT. Possuindo implementações disponíveis para uma variedade de plataformas e linguagens de programação, o cliente Paho facilitará a implementação do federador, providenciando a conexão da aplicação com o *broker* federado.

4.1.2 Eclipse Mosquitto

Mosquitto (5) é um *broker* MQTT de implementação *single-threaded*, não escalável e de código fonte aberto desenvolvido pela Eclipse Fundation.

4.1.3 MQTT *broker latency measure tool*

MQTT *broker latency measure tool* (13) é uma ferramenta escrita na linguagem Go para mensurar da latência do *broker* no envio das mensagens. Possui diversas configurações para execução, como QoS dos publicadores e assinantes, tamanho da mensagem, número de clientes e número de mensagens publicadas por cada cliente. Além da latência, a ferramenta ainda é capaz de computar outras métricas como taxa de publicação e taxa de sucesso no envio das mensagens. Essa ferramente foi projetada para a utilização nos cenários mais comuns de aplicação do MQTT onde se é utilizado um único *broker*. Dessa forma, tivemos que fazer alterações internas na ferramenta, possibilitando a configuração de assinantes e publicadores em *brokers* distinto, pois este é um requisito necessário para podermos coletar métricas de uma federação de *brokers*.

4.2 TÓPICOS

Em nossa implementação tratamos como publicações federadas (i.e., publicações que serão roteadas para outros *brokers* com assinantes locais) as publicações feitas em tópicos

que tenham primeiro nível igual a “federated”, ou seja, os tópicos de publicações federadas seguem o formato “federated/X”, onde X é um tópico federado. O federador assina todos os tópicos federados do seu *broker* hospedeiro através da utilização de um curinga multinível: “federated/#”.

Além dos tópicos federados, também são usados pelo federador quatro tópicos de controle que seguem uma estrutura parecida: os primeiro níveis do tópico identificam qual mensagem será trafegada por ele, já o último nível identifica a qual tópico federado a mensagem é relacionada. Os tópicos de controle a quais o federador assina são os seguintes:

- **federator/core_ann/#**: Destinado a anúncios de núcleo.
- **federator/memb_ann/#**: Destinado a anúncios de membros de malha.
- **federator/routing/#**: Destinado a publicações federadas sendo roteadas.
- **federator/beacon/#**: Para o recebimento de *beacons* informando a existência de assinantes locais.

Ao utilizar um curinga multinível substituindo o ultimo nível dos tópicos de controle o federador passa a receber mensagens de controle referente a todos os tópico federados. Exemplo: uma publicação recebida no tópico “federator/core_ann/sensor” indicaria um anúncio de núcleo referente ao tópico federado “sensor”.

4.3 ARQUITETURA DA APLICAÇÃO

A implementação[1] do federador foi realizada na linguagem de programação Rust 2021 com a utilização do sistema de execução (*runtime*) Tokio (11) e da biblioteca de cliente MQTT Paho. A sua arquitetura, como ilustra a Figura 6, tem como elementos principais o *dispatcher*, múltiplas filas de mensagens e os múltiplos trabalhadores de tópicos. Tanto o *dispatcher* quanto os vários trabalhadores de tópicos consistem em unidades de execução assíncronas chamadas de *tasks*: semelhantes a *threads* do sistema, mas em vez de serem gerenciadas pelo escalonador do sistema, são gerenciadas pelo *runtime* Tokio. Em comparação a *threads* do sistema, *tasks* Tokio são mais leves de se criar, executar e destruir.

Nesta arquitetura, trabalhadores de tópicos são os componentes que de fato realizam todo o trabalho de um federador: fazem a gerencia de malhas e o roteamento das mensagens. Para cada tópico federado, e consequentemente para cada malha, existe um trabalhador de tópico correspondente responsável. Publicações oriundas da assinatura dos tópicos de controle e tópicos federados no *broker* hospedeiro chegam através do cliente Paho MQTT e são direcionadas para o *dispatcher*, que por sua vez identifica o tópico federado correspondente à mensagem e a encaminha para o trabalhador de tópico responsável. Por exemplo: publicações recebidas nos

[1] `https://github.com/nicolaskribas/mqtt-fed`

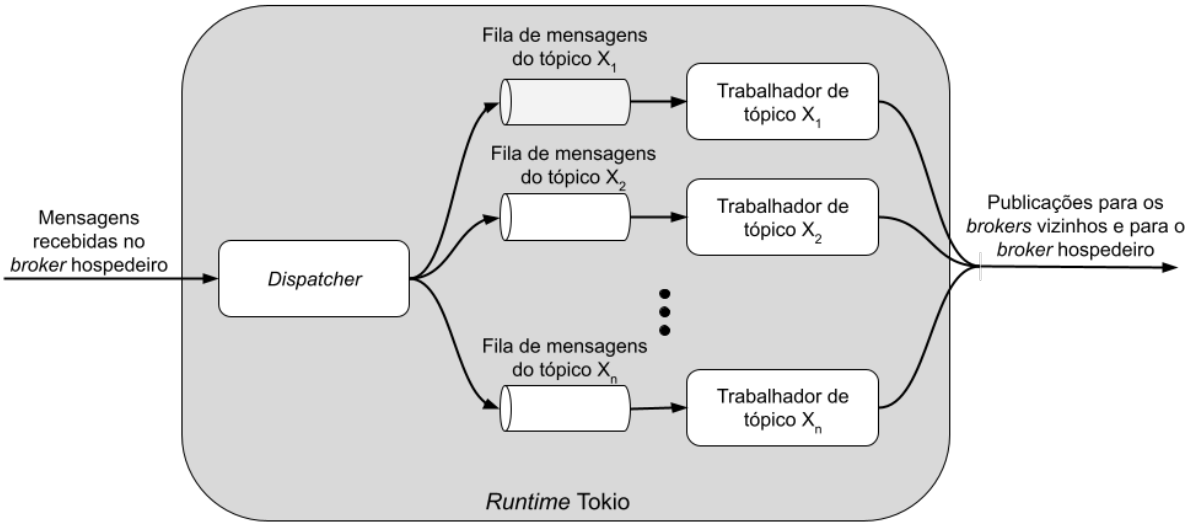

Figura 6 – Arquitetura do federador

tópicos “federated/sensor1” ou “federator/core_ann/sensor1” são processadas por um mesmo trabalhador de tópico, o qual é responsável pelo tópico federado “sensor1”.

Trabalhadores de tópicos são criados dinamicamente conforme tópicos federados são utilizados. Tópicos federados que não estão sendo utilizados, e consequentemente não possuem uma malha, não precisam de um trabalhador de tópico.

4.4 *BEACON*

Para que o federador possa realizar o seu papel ele precisa ser informado de novas publicações e assinaturas feitas no *broker* hospedeiro. Com essa finalidade, no protocolo original, utilizava-se o tópico *NEW_REGULAR_TOPIC*: um tópico onde clientes devem publicar informando toda primeira publicação ou assinatura feita em um tópico federado. Em nossa implementação o federador assina todos os tópicos federados, não havendo a necessidade de ser informado de novas primeira publicações, pois já as recebe do *broker*.

Entretanto, permanece a necessidade de clientes notificarem explicitamente suas assinaturas, pois não há outra forma com a qual o federador possa identifica-las. Para isso, clientes assinantes devem fazer a publicação de um *beacon* no tópico de *beacon* correspondente ao tópico federado ao qual ele está assinado. Por exemplo, um cliente que assina o tópico federado “sensor1” deve fazer a publicação de um *beacon* no tópico “federator/beacon/sensor1”. Um *beacon* consistem em uma mensagem vazia e deve ser publicada regularmente. O intervalo esperado entre cada *beacon* é passível de configuração pelo arquivo de configuração do federador e deve ser acordado com os clientes. Caso nenhum *beacon* seja recebido para um determinado tópico por um período superior a 3 vezes o intervalo esperado, o federador assume não haver mais nenhum assinante local e ele tomará ações baseado nessa informação, como deixar de ser membro de uma malha ou parar de se anunciar como núcleo para o tópico.

4.5 ROTEAMENTO

O trafego de publicações federadas ocorre por meio de dois conjuntos de tópicos distintos, e já mencionados: Tópicos federados (i.e., “federated/#”) e tópicos de roteamento (i.e., “federator/routing/#”). O que diferencia estes dois conjuntos é o fato de que nos tópicos federados trafegam as publicações em sua forma “crua” entre clientes locais e o federador. Já os tópicos de roteamento são de utilização exclusiva dos federadores e trafegam por eles publicações que, além do conteúdo original das publicações federadas, possuem um identificador único e um campo indicando o remetente da mesma.

Os identificadores de publicações federadas são compostos por dois valores: o ID do *broker* onde a publicação se originou; e um número de sequência, iniciado em 0 e incrementado a cada nova publicação local, que tem sua utilidade na filtragem de publicações duplicadas. Já o campo de remetente indica de qual *broker* vizinho a publicação foi roteada; essa informação nos poupa uma publicação, pois podemos ignorar esse vizinho ao repassarmos a publicação adiante, já que foi dele que a recebemos.

Ao receber uma nova publicação em um tópico federado local o federador inicia o processo de roteamento da mesma, primeiramente lhe atribuindo um identificador único e um remetente, como ilustra a figura 7, e então a enviando para os *brokers* vizinhos membros da malha (caso o *broker* atual pertença a malha) ou em direção ao núcleo (caso o *broker* não participe da malha), utilizando-se dos tópicos de roteamento para tal. A figura 8 exemplifica este processo inicial de roteamento de uma publicação local, tendo “sensor1” como tópico federado.

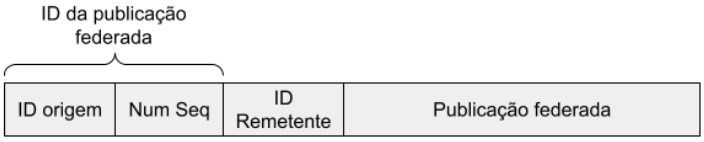

Figura 7 – Estrutura de uma publicação sendo roteada

Para publicações federadas recebidas em um tópico de roteamento, o federador deve, primeiramente, certificar-se de que não se trata de duplicatas e, para isso, deve-se manter um registro de identificadores das publicações que já foram roteadas. Em nossa implementação, esse registro é feito utilizando uma *cache* (2) cuja política de substituição é do tipo LRU (*Least Recently Used*, menos recentemente usada, em português); ou seja, novas entradas substituem entradas mais antigas, armazenando-se os identificadores das publicações federadas roteadas recentemente. O tamanho desta *cache* é passível de configuração através do arquivo de configuração do federador, pois depende da topologia utilizada.

Após verificar o identificador da publicação na *cache* e certificar-se que não se trata de uma duplicata, o federador repassará essa publicação para os vizinhos necessários, seguindo a mesma lógica do roteamento de uma publicação local. Também deverá desencapsular o conteúdo

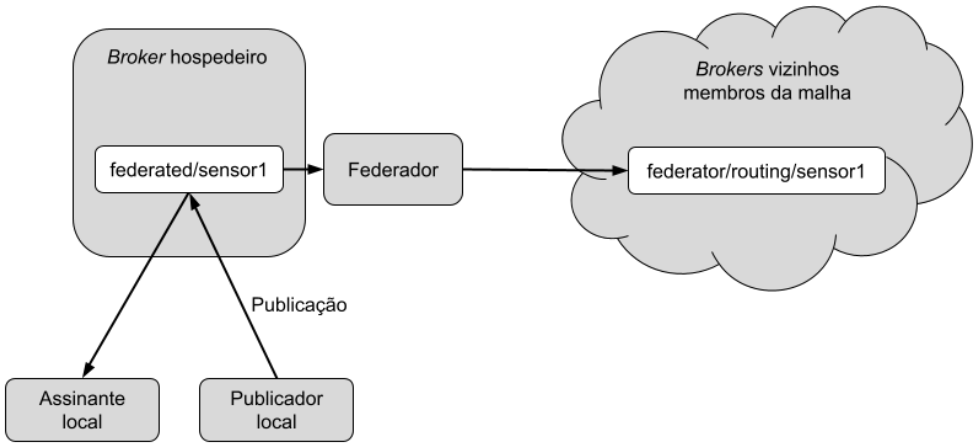

Figura 8 – Roteamento de publicação local

original e, caso tenha assinantes locais, publicar no tópico federado local correspondente. Esse processo é ilustrado na Figura 9, onde se destaca “sensor1” como exemplo de tópico federado.

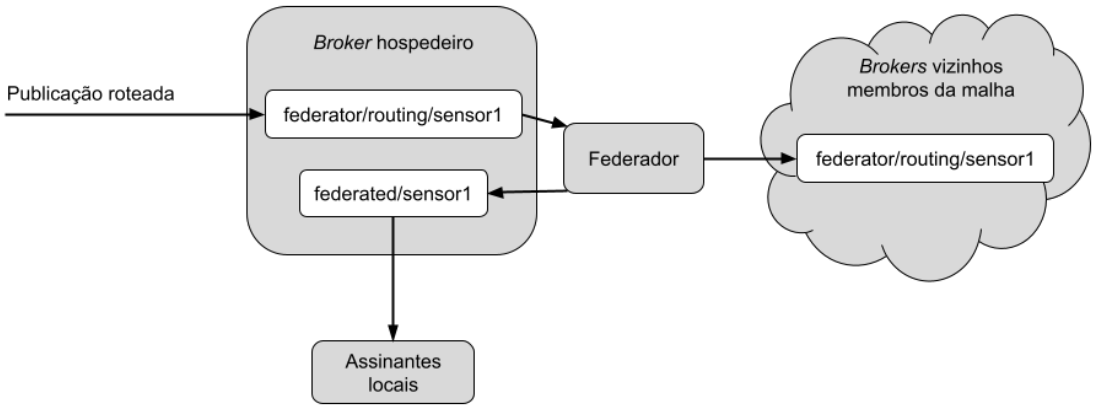

Figura 9 – Recebimento de publicação roteada

Alguém pode argumentar que por publicar nos mesmos tópicos federados que assina, o federador causaria um ciclo, recebendo suas próprias publicações e as roteando novamente com novos identificadores, causando duplicatas. Isso é resolvido em nossa implementação pois, ao assinar os tópicos federados, o federador utiliza-se da opção *No Local* que informa ao *broker* que não devemos receber nossas próprias publicações. Essa opção não está presente nas versões 3.1.1 e anteriores do MQTT, limitando a utilização de nosso federador a *brokers* que suportam a versão 5 do protocolo.

Em nossa implementação o roteamento de uma publicação feita em um *broker* não participante da malha é feito utilizando todos os pais disponíveis até o núcleo, diferente do funcionamento original em que se utiliza apenas um único caminho. Essa mudança tem como objetivo utilizar toda a redundância disponível para se fazer o roteamento.

4.6 CONFIGURAÇÕES DA APLICAÇÃO

As configurações do federador são lidas, durante sua inicialização, de um arquivo de configuração no formato TOML [2]. O local do arquivo deve ser informado no comando de execução da aplicação, caso contrário o federador lê, por padrão, o arquivo *mqtt-fed.toml* em busca das configurações.

Parâmetros que devem estar contidos no arquivo de configuração:

- *redundancy*: Inteiro positivo que designa a redundância das malhas criadas;
- *core_ann_interval*: Inteiro positivo que designa o intervalo, em segundos, entre anúncios de núcleos subsequentes.
- *beacon_interval*: Inteiro positivo que designa o intervalo, em segundos, esperado entre os *beacons* vindos de assinantes locais.
- *cache_size*: Inteiro positivo que designa o tamanho máximo das *caches* utilizadas para filtrar duplicatas de cada tópico federado.
- *host*: Tabela que representa o *broker* hospedeiro, deve conter o identificador e URI do mesmo.
- *neighbors*: Array de tabelas, cada elemento desse array deve conter o identificador e o URI de um *broker* vizinho.

A Listagem 4.1 contém um exemplo de arquivo de configuração de um federador associado ao *broker* com identificador 1 e que tem como vizinhos os *brokers* de identificadores 2 e 4.

[2] `https://toml.io/en/v1.0.0`

Listagem 4.1 – Exemplo de configuração do federador

```
redundancy = 2
core_ann_interval = 10
beacon_interval = 5
cache_size = 5000

[host]
id = 1
uri = "tcp://localhost:1881"

[[neighbors]]
id = 2
uri = "tcp://localhost:1882"

[[neighbors]]
id = 4
uri = "tcp://localhost:1884"
```

5 ESTUDO DE CASO

5.1 CENÁRIOS DE AVALIAÇÃO

Os cenários de avaliação utilizados são uma tentativa de replicar os mesmos utilizados por Spohn (8). Sendo assim, a topologia conta com nove *brokers* federados dispostos em uma grade 3×3, com redundância de malha de valor dois. Nessa topologia foram dispostos dois assinantes, o primeiro no nodo 3 e o segundo no nodo 8. A malha resultante pode ser observada na Figura 10. Cada nodo da federação é composto por um *broker* Mosquitto executando em um contêiner Docker, em conjunto com uma instância de federador. As configurações usadas que são comuns entre os federadores estão presentes na Tabela 1.

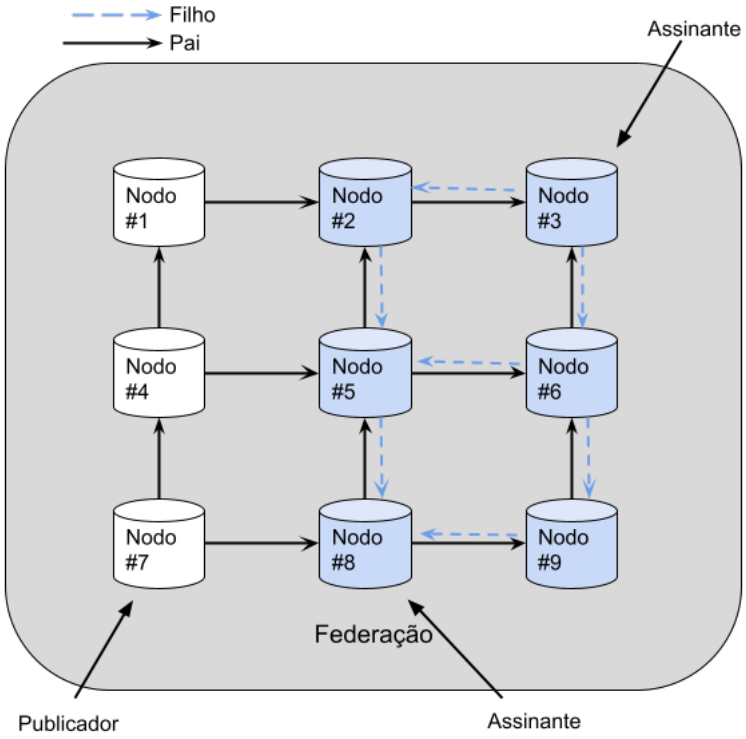

Figura 10 – Cenário de avaliação

Tabela 1 – Configurações comuns entre os federadores

Parâmetro	Valor usado
Intervalo entre anúncios de núcleo	10 s
Intervalo esperado entre *beacons*	5 s
Tamanho da cache	5000

Um publicador foi acionado no nodo 7, a um salto de distância do nodo 8 e a quatro saltos do nodo 3. O publicador foi configurado de duas maneiras: fazendo 500 e 1000 publicações, cada uma delas com um tamanho de 64 bytes. Coletamos como métrica o atraso entre uma mensagem ser publicada no nodo 7 e sua entrega nos assinantes nos nodos 3 e 8.

Para avaliar um cenário de único *broker* e podermos fazer as devidas comparações com a abordagem federada, executamos ambos os assinantes em um mesmo *nodo*: primeiramente com ambos no nodo 3 e então com ambos no nodo 8. Utilizamos o mesmo padrão de publicador, com 500 e 1000 publicações sendo realizadas no nodo 7.

Cada um dos cenários foi executado 10 vezes, coletando-se as médias dos resultados das execuções. Os testes foram executados em uma maquina local que dispunha de 16GB de memória RAM e um processador *AMD Ryzen™ 5 1500X 3,5 GHz*.

5.2 RESULTADOS

Na Tabela 2 estão dispostos os resultados coletadas no cenário de federação. Como esperado, o assinante do *broker* 3, que se encontra mais distante do publicador, mostrou valores de latência maiores do que o assinante do *broker* 8, que se encontra mais perto do publicador. Observa-se também que a quantidade de publicações não afetou significativamente a latência no *broker* 3. O mesmo não pode ser dito do assinante do *broker* 8, suas publicações tiveram uma latência maior com o aumento do número de publicações, o que era esperado.

Tabela 2 – Solução federada: Atraso na publicação das mensagens

Publicações	Assinante no *broker* 3	Assinante no *broker* 8
500	17,21±9,17 ms	9.51±6,58 ms
1000	16,71±9,53 ms	12,35±10,70 ms

Os cenários de *broker* único tem seus resultados demonstrados na Tabela 3. Novamente observa-se uma latência relacionada com a distância entre publicador e assinante.

No cenário onde ambos assinantes localizam-se no nodo 3, nota-se um aumento na latência agregada em comparação com a solução federada. Isso se deve pelo fato do *broker* ter que lidar com o dobro da carga na média, pois possui dois assinantes. Já no cenário onde ambos assinantes estão no nodo 8 a constatação é a contrária, pois houve uma redução significativa de latência em comparação com o cenário federado. Uma explicação possível para essa diferença entre os dois cenários envolve o *overhead* gerado pelo roteamento das mensagens. Tanto no cenário federado quanto no cenário de um único *broker* no nodo 3, o roteamento de mensagens é feito utilizando todos os pais que a redundância permite, resultando em uma sobrecarga no roteamento. Já no cenário de único *broker* no nodo 8 o *overhead* é mínimo, pois estando a um salto do publicador, o único caminho (e único pai) pelo qual as mensagens são roteadas é através do envio direto ao *broker* dos assinantes.

Tabela 3 – Solução centralizada: Atraso na publicação das mensagens

Publicações	*Broker* no nodo 3		*Broker* no nodo 8	
	Sub 1	Sub 2	Sub 1	Sub 2
500	19,81±16,21 ms	18,80±13,05 ms	3,33±4,35 ms	3,34±4,35 ms
1000	17,66±15,47 ms	17,68±15,49 ms	5,04±7,05 ms	5,10±7,12 ms

6 CONCLUSÃO

Com o crescimento e uma maior adoção da Internet das Coisas, vê-se a necessidade de protocolos de transmissão de mensagens escaláveis capazes de suprir a necessidades de comunicação entre os elementos desse paradigma. Uma das alternativas é a federação de *brokers* MQTT.

O presente trabalho limitou-se à uma análise inicial das capacidades de escalabilidade de uma federação de *brokers*. Para realizar a federação foi implementado um federador: aplicação que, associada a um *broker*, dá as capacidades de federação. Além de servir ao propósito de avaliarmos uma federação no presente trabalho, o federador implementado pode servir de ferramenta a toda a comunidade interessada em explorar a abordagem de federação.

Os resultados obtidos no estudo de caso mostraram que a federação pode trazer benefícios de desempenho em comparação com o uso de *brokers* centralizados. Também serviram pra ilustrar os *overheads* que o protocolo de federação apresenta em certas topologias.

6.1 TRABALHOS FUTUROS

Como próximos passos no estudo da federação de *brokers*, pode-se citar uma comparação mais direta com outras abordagens de escalabilidade do MQTT (e.g., clusterização de *brokers*). Também necessita-se de trabalhos que explorem as capacidades de tolerância a falhas do protocolo de federação.

REFERÊNCIAS

1 ECLIPSE Paho. Eclipse Foundation. Disponível em: <https://www.eclipse.org/paho/>. Acesso em: 5 out. 2021.

2 FROELICH, Jerome. **An implementation of a LRU cache**. Disponível em: <https://github.com/jeromefroe/lru-rs>.

3 AL-FUQAHA, Ala et al. Internet of things: A survey on enabling technologies, protocols, and applications. **IEEE communications surveys & tutorials**, IEEE, v. 17, n. 4, p. 2347–2376, 2015.

4 LEE, In; LEE, Kyoochun. The Internet of Things (IoT): Applications, investments, and challenges for enterprises. **Business Horizons**, Elsevier, v. 58, n. 4, p. 431–440, 2015.

5 LIGHT, Roger A. Mosquitto: server and client implementation of the MQTT protocol. **Journal of Open Source Software**, v. 2, n. 13, p. 265, 2017.

6 MISHRA, Biswajeeban; KERTESZ, Attila. The use of MQTT in M2M and IoT systems: A survey. **IEEE Access**, IEEE, v. 8, p. 201071–201086, 2020.

7 PIPATSAKULROJ, Wiriyang; VISOOTTIVISETH, Vasaka; TAKANO, Ryousei. mumq: A lightweight and scalable mqtt broker. In: IEEE. 2017 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN). [S.l.: s.n.], 2017. p. 1–6.

8 SPOHN, Marco Aurélio. An endogenous and self-organizing approach for the federation of autonomous MQTT brokers, 2021.

9 _____. Publish, subscribe, and federate!, 2020.

10 STANDARD, OASIS. MQTT Version 5.0. **Retrieved June**, v. 22, p. 2020, 2019.

11 TOKIO API Docs. Tokio. Disponível em: <https://docs.rs/tokio/latest/tokio/>. Acesso em: 16 mar. 2022.

12 TAM Forecasts. Transforma Insights. Disponível em: <https://transformainsights.com/research/forecast/highlights>. Acesso em: 5 out. 2021.

13 ZHANG XIANG, Jianhui. **MQTT broker latency measure tool**. Disponível em: <https://github.com/hui6075/mqtt-bm-latency>. Acesso em: 5 out. 2021.