Aquisição de Dados Escalável e Ciente da Aplicação para Gêmeos Digitais

Rafael Trevisan[1], **Francisco Paiva Knebel**[1], **Juliano Araújo Wickboldt**[1], **Mara Abel**[1]

[1]Instituto de Informática – Universidade Federal do Rio Grande do Sul (UFRGS)
Porto Alegre – RS – Brasil

{rafael.trevisan,francisco.knebel,jwickboldt,marabel}@inf.ufrgs.br

Abstract. *Digital Twin is the name given to a type of system that has full integration between a physical part and a digital part updated in real time. In general, they are large-scale systems with intense flow of data between their parts (physical and digital). To perform data acquisition in this type of system, protocols such as MQTT are commonly used. This work recognizes the limitations of this protocol when using high traffic conditions and explores alternatives to increase data flow while minimizing message loss and resource usage. For this, the native cluster mode of MQTT broker implementations is used, adding a component to the system architecture which we call "Cluster Manager". This component was developed with the objective of monitoring the health of a cluster of brokers and adjusting the composition of the cluster to the message flow. The results of the implementation of this component helped to determine loss margins and latencies in sending critical messages, using different strategies to scale the cluster under representative workloads of Digital Twins applications. It has also been determined that using very sensitive parameters to scale the system causes more nodes than necessary to be instantiated and resources wasted.*

Resumo. *Gêmeo Digital é o nome dado a um tipo de sistema que possui total integração entre uma parte física e uma parte digital atualizada em tempo real. No geral, são sistemas de grande escala com intenso fluxo de dados entre as suas apartes (física e digital). Para realizar a aquisição de dados nesse tipo de sistema, utiliza-se comumente protocolos como MQTT. Este trabalho reconhece as limitações desse protocolo quando utilizado condições de tráfego intenso e explora alternativas para aumentar o fluxo de dados minimizando a perda de mensagens e o uso de recursos. Para isso é usado o modo cluster nativo de implementações de brokers MQTT, adicionando um componente à arquitetura do sistema ao qual chamamos de "Cluster Manager". Este componente foi desenvolvido com o objetivo de monitorar a saúde de um cluster de brokers e ajustar a composição do cluster ao fluxo de mensagens. Os resultados da implantação desse componente ajudaram a determinar margens de perda e latências no envio de mensagens críticas, utilizando diferentes estratégias para escalar o cluster sob cargas de trabalho representativas de aplicações de Gêmeos Digitais. Também foi determinado que o uso de parâmetros muito sensíveis para escalar o sistema faz com que mais nodos que o necessário sejam instanciados e recursos desperdiçados.*

1. Introdução

O conceito de Digital Twin (DT ou Gêmeo Digital) tem recebido muita tração nos últimos anos. Criado em 2002, ele define um conjunto de métodos e mecanismos com objetivo de replicar um objeto e suas características físicas em um espaço virtual correspondente e atualizá-lo em tempo real [Grieves 2016]. Através da análise de dados históricos, modelos de simulação e das informações recebidas em tempo real, um DT é capaz de fazer previsões sobre seu próprio estado no futuro, dar recomendações de uso e alertar os usuários sobre problemas em potencial, para que eles possam ser abordados de maneira proativa [Boschert and Rosen 2016]. Para ser capaz de receber e processar dados em tempo real, é preciso se assegurar que o seu sistema de aquisição de dados seja eficiente e seguro, tendo em vista que a entrega oportuna e precisa dos dados irá influenciar diretamente as previsões feitas pelo Gêmeo Digital.

Para aquisição de dados em sistemas como DTs existe uma gama de protocolos e arquiteturas de comunicação possíveis de serem adotados, como Message Queue Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), bem como HTTP e Constrained Application Protocol (CoAP). Nesse trabalho, escolhemos utilizar o MQTT por se tratar de um protocolo leve, econômico e compatível com o uso de aparelhos de baixa capacidade computacional, como sensores, especialmente em ambientes com dispositivos conectados de forma intermitente à rede [Banks et al. 2019, Cunha and Batista 2021]. Outros protocolos de comunicação de dados como CoAP, HTTP e AMQP são alternativas possíveis, porém o MQTT foi escolhido por apresentar uma menor latência, exigir menos da rede e ser em geral mais seguro que os outros protocolos listados, além de sua popularidade em sistemas IoT já existentes [Human et al. 2021]. O MQTT funciona através de um mecanismo de *publish*/*subscribe* regulado por um mediador (*broker*). Nesse mecanismo, primeiramente, os clientes consumidores de dados (*subscribers*) registram os tópicos de seu interesse no *broker*. Em seguida, os clientes fornecedores de dados (*publishers*) enviam mensagens ao *broker*, com um certo tópico. O *broker* então se torna responsável por encaminhar as mensagens recebidas à qualquer cliente que tenha se inscrito naquele tópico.

Quando modelamos um objeto complexo em larga escala, é previsto um fluxo de dados muito grande entre os sensores que medem as características do objeto e o Gêmeo Digital. No mecanismo de comunicação *publish*/*subscribe* utilizado pelo protocolo MQTT, a agilidade para o envio de mensagens depende diretamente do desempenho do *broker*, uma vez que ele centraliza toda a comunicação. Para evitar que esse elemento se torne um gargalo, é preciso se assegurar da sua saúde e eficiência. Caso o *broker* esteja sobrecarregado, ele irá enfileirar as mensagens que não foi capaz de enviar e resolverá essa fila sequencialmente, gerando atrasos no recebimento das mensagens [Trevisan et al. 2020]. Para que possamos garantir a qualidade das análises e efetividade das ações feitas por um Gêmeo Digital, a aquisição de dados deve ser projetada como um sistema de tempo real, garantindo que que a parte virtual possa refletir o seu equivalente real de forma precisa e oportuna.

Para evitar que um *broker* MQTT fique sobrecarregado, algumas implementações de *broker* MQTT possuem um modo *cluster*, onde diversas instâncias de *broker* dividem a carga que recebem dos clientes (*publishers* e *subscribers*) entre si. A simples existência de um modo *cluster* não é suficiente para resolver o problema de sobrecarga

do *broker*. A fim de entregar a melhor solução de aquisição de dados para cada aplicação, ainda se faz necessário dimensionar adequadamente o *cluster* (*e.g.*, quantidade de nodos), alocar e disponibilizar os recursos de computação e comunicação apropriados (*e.g.*, CPU, memória, etc.), ajustar parâmetros de configuração (*e.g.*, tamanho das filas, priorização de mensagens) e posicionar os nodos do cluster adequadamente sobre a infraestrutura subjacente. Nesse trabalho, primeiramente, analisamos o comportamento de um *cluster* de *brokers* MQTT como mecanismo para aquisição de dados considerando cargas de trabalho (quantidade de clientes, volume e frequência de mensagens) típicos de aplicações de Gêmeos Digitais. Posteriormente, propomos a utilização de um "escalador automático" ciente dos parâmetros internos de configuração do *cluster* de *brokers*, chamado *MQTT Broker Manager*. Esse escalador que verifica o status de um *cluster* de *brokers* e é capaz de gerenciar o *cluster* adicionando ou removendo nodos dinamicamente para lidar com variações nos fluxos de mensagens. Através de experimentos realizados pode-se observar o comportamento do *cluster* em diferentes situações e avaliar quais seriam os impactos no funcionamento e nos requisitos de tempo real de um DT. Observando os resultados é possível também sugerir quais são os parâmetros mais adequados para justificar a expansão do *cluster*, além de verificar os impactos do uso dessa ferramenta em uma arquitetura análoga a de um Gêmeo Digital.

O restante desse trabalho está estruturado da seguinte forma. A Seção 2 discute os trabalhos relacionados. Na Seção 3 a arquitetura de um Gêmeo Digital e seus principais componentes, bem como os perfis de cargas de trabalho utilizados. Na Seção 4 são apresentados alguns experimentos preliminares e também é detalhado o MQTT Cluster Manager. Os resultados são discutidos na Seção 5 e a Seção 6 relata as conclusões obtidas através desses resultados.

2. Trabalhos Relacionados

Gêmeos Digitais surgiram do processo de *Product Lifecycle Management* (PLM), proposto como uma representação não estática de um produto durante todo o seu ciclo de vida desde a sua fabricação até o fim de seu uso [Grieves 2016]. Desde a criação deste conceito, já estava prevista a separação entre espaços reais e espaços virtuais e o fluxo de dados indo do espaço real para o virtual. Com o tempo, a ideia de PLM evoluiu para o conceito mais moderno de gêmeo digital, mantendo as suas principais ideias, mas incluindo novas propostas.

A expansão de dispositivos sensores de baixo custo, devido à Internet das Coisas (*Internet of Things*, IoT) é um dos principais fatores motores ao desenvolvimento de gêmeos digitais, sendo uma solução puramente digital e de fácil implementação, não precisando uma duplicação física do sistema, e sim apenas uma definição dos requisitos que o compõem e alimentação de dados para seus algoritmos. Latência é um problema de muita preocupação com a comunicação IoT, especialmente no contexto de aplicações industriais, e com a maior adaptação e implementação, serviços de baixa latência se tornarão cada vez mais importantes [Ferrari et al. 2017]. Com o crescimento da demanda e a criação de comunicação em larga escala, a quantidade de tráfego na rede pode ser um limitante para a resultante inovação tecnológica vinculada à gêmeos digitais, onde fatores como escalabilidade do sistema precisam ser consideradas na sua implementação.

Para resolver isso de forma que não haja prejuízo na comunicação, através da

perda de mensagens, o funcionamento do *broker* em modo *cluster*, ou seja, múltiplos *brokers* ligados entre si e trabalhando em conjunto, permite obter melhor desempenho (maior poder de processamento), disponibilidade (sistema resistente a falhas, redundante), e efetuando balanceamento de carga (distribuição equilibrada do processamento).

O uso de MQTT para efetuar a comunicação de DTs também é um conceito bem explorado na literatura. Koziolek *et al.* apresentam um estudo comparativo sobre o uso de diferentes *brokers* MQTT, com o objetivo de executar em cenários de alta escalabilidade e redundância [Koziolek et al. 2020]. Os autores, através da experimentação efetuada nos *brokers*, descobriram que dentre as opções disponíveis no mercado, o EMQ X possui a maior vazão de mensagens. Thean *et al.* exploram o uso de um *cluster* MQTT [Thean et al. 2019], tirando conclusões sobre a sua implementação em ambientes de computação em borda. Neste trabalho, é demonstrado que a latência média *end-to-end* para esse tipo de uso está abaixo de 10ms, sendo possível utilizá-la em contextos que necessitam baixa latência, mas com baixa variação, devido à resiliência adicionada pelo *cluster*.

Outros trabalhos que efetuaram análises de desempenho do *broker* MQTT apontaram diferenças, seja entre as diferenças entre as implemnetações de *broker* ou com o uso de diferentes protocolos [Mishra 2018]. Entretanto, um estudo mais profundo ainda é possível sobre outros aspectos que podem influenciar a comunicação de sensores e máquinas instanciando gêmeos digitais, como a distribuição da carga gerada pelo fluxo de mensagens recebidas entre múltiplos *brokers*, com objetivo de maior escalabilidade do sistema.

3. Arquitetura do Gêmeo Digital

A arquitetura utilizada neste trabalho foi organizada de uma forma simplificada para permitir uma análise mais aprofundada dos componentes que integram o sistema de aquisição de dados de um Gêmeo Digital [Damjanovic-Behrendt and Behrendt 2019]. Outros componentes para processamento, armazenamento e simulações não serão analisados neste trabalho e podem ser incorporados em trabalhos futuros. Para reproduzir o comportamento da infraestrutura de comunicação para aquisição de dados em Gêmeos Digitais foram utilizados *brokers* MQTT e pares clientes-Gêmeo capazes de simular a geração e aquisição de dados neste contexto. Os componentes da arquitetura são apresentados na Figura 1 e detalhados a seguir.

3.1. *Cluster* MQTT

Como protocolo de aquisição de dados foi escolhido o MQTT, e o *broker* utilizado foi o EMQ X na versão 4.3.8. Esta implementação foi escolhida por sua versatilidade, escalabilidade e alta performance para sistemas em tempo real [EMQ Inc. 2022] e facilidade para criação de *clusters* distribuídos. Para criar o *cluster*, foi utilizado um servidor *etcd*[1], através do qual as diferentes instâncias individuais de *broker*, chamadas de nodos, se associam. Quando associadas dessa maneira, essas diferentes instâncias compartilham estruturas de dados de roteamento de mensagens e permitem que diferentes nodos do *cluster* de *brokers* MQTT compartilhem mensagens entre si.

[1]Serviço usado como um banco de dados distribuído, de alta disponibilidade, baseado em chave-valor. Ele resolve o problema de serviços membros de um *cluster* se encontrarem e estabelecerem comunicação. Disponível em `https://etcd.io/`.

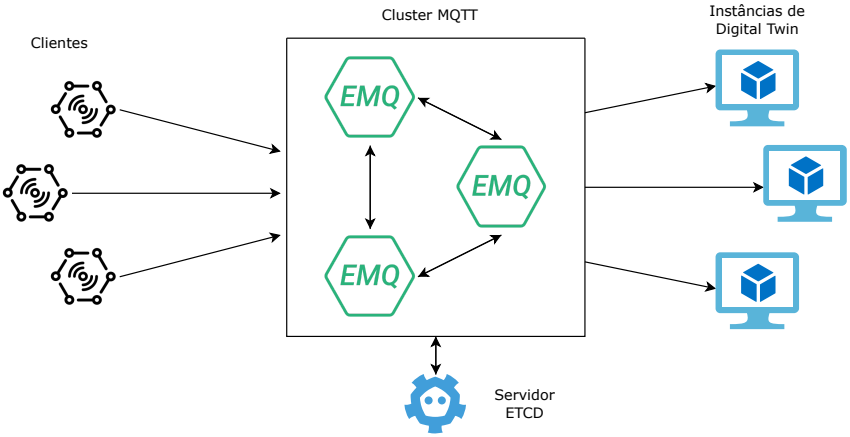


Figura 1. Arquitetura do sistema de aquisição de dados para Gêmeos Digitais

Um *cluster* de *brokers* pode ser iniciado com um número arbitrário de nodos e também é possível que nodos entrem e saiam dinamicamente no *cluster*. Com o ingresso de novos nodos em um *cluster* é esperado que a capacidade de processamento de mensagens aumente (e com a saída espera-se o efeito oposto). Novos nodos ingressam no cluster sem clientes conectados (*publishers* ou *subscribers*), sendo assim, para que eles participem do sistema e dividam a carga de trabalho ainda é preciso que algum mecanismo de balanceamento seja utilizado para redistribuir a carga existente entre os nodos ingressantes.

Além da formação do *cluster*, algumas configurações por nodo podem ser utilizadas para afetar o desempenho do sistema de aquisição de dados como um todo. Por exemplo, é possível definir níveis de priorização de mensagens por tópico, tamanho máximo da fila de mensagens, número máximo de mensagens em trânsito (*i.e.*, mensagens que ainda aguardam confirmação de recebimento), parâmetros estes que têm efeito no desempenho global do sistema. É importante observar que mesmo ampliando esses valores máximos (tamanho de fila e mensagens em trânsito) o *broker* ainda é limitado pelos recursos de CPU e memória da máquina onde ele está sendo executado. Portanto, é possível perceber que ajustar um sistema de aquisição de dados baseado em um cluster de *brokers* MQTT para obter agilidade e eficiência na comunicação não é uma tarefa trivial.

3.2. Par Clientes-Gêmeo Digital

Neste trabalho foram utilizadas implementações de clientes que enviam mensagens de maneira controlada. Nesta implementação cria-se um número configurável de *threads*, em um intervalo também configurável, que se conectam individualmente com o *cluster* e enviam mensagens ao longo do tempo, simulando diversos aparelhos (*e.g.*, sensores).Ao decorrer do tempo mais *threads* são geradas e se conectam ao *cluster*, até o cliente atingir uma taxa máxima de envio de mensagens, que decresce até o final da execução do programa.Cada um dos clientes utilizados foi configurado para simular uma classe de serviço que representa a carga de trabalho de diferentes tipos de Gêmeo Digital, ajustando as janelas de transferência de dados, tamanho das mensagens e QoS, conforme a Tabela 1. Os parâmetros da referida tabela e as classes utilizadas foram inspirados no QoS Class Identifier (QCI) do 3GPP [3GPP 2021] e no trabalho de Maaloul *et al.* [Maaloul et al. 2021].

Na Tabela 1, apresentamos cinco classes de serviço diferentes para aplicações de missão crítica (alta prioridade), automação industrial com dois tamanhos de mensagens, aplicações veiculares (V2X) e aplicações baseadas em vídeo (e.g, reconhecimento de objetos). A janela de dados (*Janela (ms)*) determina qual o intervalo entre o envio de mensagens de cada um dos serviços. A coluna *Mensagem (bytes)* determina qual o tamanho dos pacotes que são enviados por esse tipo de cliente. Quanto menor o valor definido na coluna *Prioridade* mais prioritárias são as mensagens enviadas pelo tipo de cliente. A coluna de *Latência Máx (ms)* indica qual a latência máxima aceitável entre o envio de uma mensagem e o recebimento dela para cada serviço. Cada mensagem enviada por um cliente de qualquer tipo contém (em sua *payload*) o valor do *timestamp* do momento em que ela foi criada para que esse valor possa ser comparado com o momento onde ela foi recebida pela instância de Gêmeo Digital.

Serviço	**Janela (ms)**	**Mensagem (bytes)**	**Prioridade**	**Latência Máx (ms)**
Mission Critical Application	500	64	0.5	75
Discrete Automation (small)	2000	255	1.9	10
Discrete Automation (big)	2000	1354	2.2	10
V2X	100	64	3	50
Video Feed	66	52 k	5	300

Tabela 1. Características dos serviços.

A instância de Gêmeo Digital é um protótipo da parte virtual de um gêmeo[2] que implementa os mecanismos necessários para aquisição de dados oriundos da parte física do DT. A instância de Gêmeo Digital se inscreve nos seus tópicos de interesse e registra as mensagens recebidas em disco, salvando também o momento onde ela foi recebida. Dessa forma é possível comparar o momento em que uma mensagem foi enviada, contido em sua *payload*, com o momento em que ela foi recebida. Neste trabalho não consideramos nenhum tipo de processamento das mensagens recebidas nem envio no sentido inverso (da instância de Gêmeo Digital para o cliente), apesar de serem operações possíveis e necessárias, ficam, por hora, fora do escopo do trabalho.

4. Implementações

Conforme exposto anteriormente, configurar um sistema de aquisição de dados baseado em um cluster de *brokers* MQTT a fim de obter uma comunicação ágil e eficiente não é uma tarefa trivial. Considerando isto, na Subseção 4.1 discutimos alguns resultados de uma série de experimentos preliminares realizados no intuito de melhor identificar os impactos da configuração do em *cluster* para o desempenho do sistema de aquisição de dados. Posteriormente, na Subseção 4.2 detalhamos a concepção do Cluster Manager proposto neste trabalho para configurar dinamicamente o *cluster* de *brokers* MQTT.

4.1. Implementação Preliminar

A fim de melhor compreender a dinâmica de funcionamento dos componentes do sistema de aquisição de dados baseado no *cluster* de *brokers* MQTT foram realizados uma série

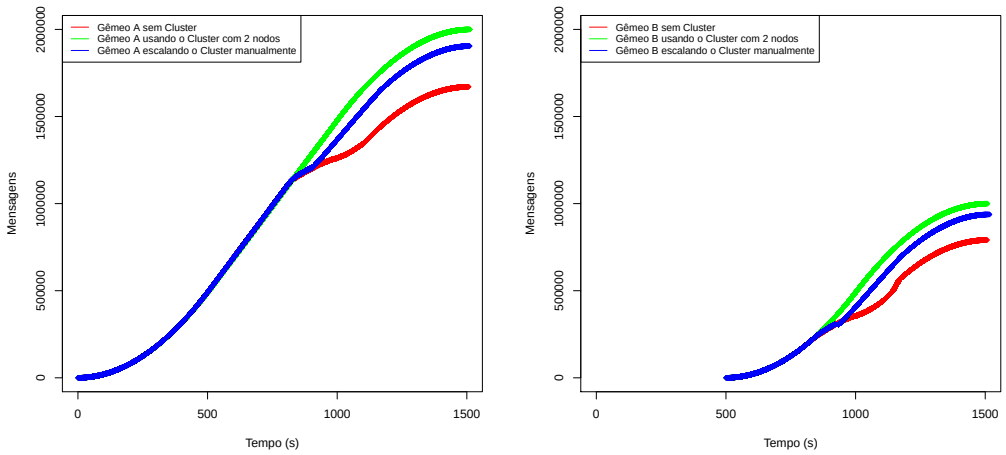
[2]Código aberto disponível em `https://github.com/Open-Digital-Twin`.

de experimentos preliminares. Como ambiente de experimentação foi utilizado um servidor com processador Intel Xeon E5-2420 (com 6 núcleos físicos e 12 *threads*) e 32 GB de RAM. Nesse servidor foram instanciadas máquinas virtuais, conectadas a uma rede local também virtual estabelecida através de uma Linux Bridge, e com duas configurações de recursos computacionais: Máquinas virtuais do tipo 1, com 2 GB de memória e 1 vCPU e máquinas virtuais do tipo 2, com 4 GB de memória e 2 vCPUs. Os diferentes componentes do sistema utilizados foram lançados através do uso de *containers* Docker. Para instanciar esses componentes nessas máquinas virtuais foi utilizado o *Docker Swarm*, uma forma de nativamente gerenciar um grupo de *docker engines* e distribuir eficientemente *containers* em diferentes máquinas.

Considerando esse ambiente, para criar um primeiro cenário com um perfil de tráfego estável foi conduzido um experimento que simulava diversos clientes de apenas uma das classes (*i.e.* V2X) apresentadas na Tabela 1. Nesse primeiro experimento, cada cliente da classe V2X se comunica com uma instância de Gêmeo Digital (*Gêmeos A*), ambos instanciados em uma máquina do tipo 2, até um máximo de 100 conexões simultâneas enviando um total de 2 000 000 de mensagens, através de um *broker*, posicionado em uma máquina do tipo 1. Durante o experimento (aos 500 segundos), o fluxo de mensagens é ampliado pois um novo par cliente MQTT-instância de Gêmeo Digital (*Gêmeos B*) é criado e posicionado em outra máquina do tipo 2 para enviar mais 1 000 000 de mensagens e até 50 conexões simultâneas.

Através do monitoramento da comunicação entre esses componentes, é possível encontrar limitações quando é empregando apenas um *broker*. Analisando a frequência do recebimento de mensagens nos Gêmeos A e B, apresentados nas Figuras 2a e 2b respectivamente, é possível perceber que com essas configurações o *broker* está sobrecarregado apresenta taxas de entrega de mensagens inconsistentes (linha vermelha). Além disso, das 3 000 000 de mensagens enviadas para as duas instâncias de gêmeos apenas 2 461 760 foram recebidas, fazendo com que este cenário apresente uma perda de 18% das mensagens. Essa perda ocorre, pois o único *broker* utilizado nesse cenário excede o tamanho máximo da sua fila interna de mensagens, descartando novas mensagens que estejam chegando dos clientes.

Quando o mesmo experimento é realizado em um novo cenário com dois *brokers* MQTT operando no modo *cluster*, em máquina virtuais tipo 1 diferentes, desde o início do experimento, se encontram taxas de entrega bem mais estáveis (Figuras 2a e 2b, linha verde). Dessa forma, é possível evitar as perdas e manter a taxa de entrega de mensagens constante. Porém, vale destacar que foi necessário dedicar uma máquina virtual a hospedar um *broker* MQTT extra, mesmo não havendo essa necessidade nos primeiros e nos últimos minutos de execução do experimento. Considerando esse problema, realizamos uma nova rodada do experimento em um terceiro cenário onde adicionamos manualmente um nodo ao cluster aos próximo dos 800 segundos do experimento (momento em que as perdas ocorreram no cenário com um *broker* apenas). Nesse cenário, o *cluster* leva alguns segundos para realizar o ingresso do novo nodo, mas depois que o ingresso ocorre no restante do experimento o fluxo de entrega de mensagens segue estável até o fim e a quantidade menagens perdidas fica na faixa dos 5% (linha azul), resultado que apresenta uma melhora em relação ao cenário com um único *broker* e que consome menos recursos do que o cenário com dois nodos ao longo de todo o experimento. Obviamente, não é



(a) Mensagens entregues ao Gêmeo A **(b) Mensagens entregues ao Gêmeo B**

Figura 2. Ilustração das três variações do experimento, para o Gêmeo A e B.

interessante monitorar e configurar manualmente um sistema dinâmico como este. É por esse motivo que introduzimos o Cluster Manager detalhado a seguir.

4.2. Cluster Manager

O *Cluster Manager* é um componente que adicionamos ao sistema de aquisição de dados que tem como objetivo verificar se um *broker* ou *cluster* está sendo capaz de processar mensagens no mesmo ritmo em que as recebe, monitorar a saúde do *cluster* MQTT, e caso necessário, adicionar novos nodos a ele. Para verificar os status do *cluster*, esse componente faz chamadas à API do *cluster* e verifica métricas importantes como o número de mensagens na fila e número de mensagens em trânsito. Esses atributos estão interligados e inspecionando eles é possível determinar o estado do fluxo de mensagens por aquele *broker*. O número de mensagens em trânsito se refere ao número de mensagens que foram enviadas mas não tiveram seu recebimento confirmado. Quando um valor máximo (configurável) de mensagens em trânsito é atingido, o *cluster* passa a enfileirar mensagens.

Inicialmente, implementamos no *Cluster Manager* uma lógica simples baseada em limites para essas métricas de desempenho do *cluster* de *brokers* (*i.e.*, tamanho da fila e número de mensagens em trânsito). Caso esses valores estejam acima dos limites configurados no *Cluster Manager*, este faz uma chamada à API do *Docker Swarm*, ampliando a quantidade de réplicas uma nova instância de *broker*, em uma máquina virtual vaga, que se junta ao *cluster* para agilizar o processamento de mensagens.

O experimento descrito na Subseção 4.1 foi realizado mais uma vez, agora utilizando o *Cluster Manager* como parte da arquitetura, gerando os resultados muito similares aos das Figuras 2a e 2b, linha azul. Como parâmetro para a expansão do *cluster* foi utilizado o tamanho da fila em 5000 mensagens, ou seja, sempre que a fila ultrapassa este valor uma réplica do *broker* é instanciada automaticamente. A saber, este valor (5000) também é o tamanho máximo da fila configurado por padrão pelo EMQ X. Para esses experimentos configuramos o tamanho máximo da fila como ilimitada, passando a controlar a expansão da fila através da lógica interna do *Cluster Manager*. Com o componente

Cluster Manager implementado na arquitetura, das 3 000 000 de mensagens 2 842 300 foram recebidas, ou seja, ocorreu uma perda de aproximadamente 6% do total. Este resultado se assemelha expansão manual realizada anteriormente, mas obviamente representa um avanço pois demonstra que o *Cluster Manager* cumpre o seu papel ao automatizar o processo.

É importante salientar que o funcionamento interno do *Cluster Manager* foi implementado inicialmente de forma bastante simples apenas como uma prova de conceito. Certamente, mecanismos mais sofisticados podem ser empregados utilizando os mesmos dados já mencionados e também uma série de outras métricas disponíveis tanto na API do *broker* quanto de outras fontes (*e.g.*, estatísticas do Docker e do Swarm), mecanismos estes que pretendemos realizar em trabalhos futuros. Com os resultados apresentados até aqui já é possível perceber que existe um potencial significativo de melhorar o sistema de aquisição de dados utilizando o *Cluster Manager*. Embora não tenha sido possível prevenir a perda de mensagens, com o decorrer do tempo existe um ganho no total de mensagens que foram preservadas. Na Seção 5, abordamos mais profundamente os fatores que podem ser utilizados como gatilhos para replicar nodos do *broker* com o objetivo de melhor compreender a influência de cada parâmetro no desempenho geral do sistema.

5. Experimentos e Discussão

5.1. Metodologia

Levando os resultados dos experimentos preliminares em consideração, realizamos mais uma série de experimentos com o intuito de simular cenários mais realísticos utilizando variadas classes de aplicações de Gêmeos Digitais e avaliando também diferentes parâmetros de configuração do Cluster Manager. Como ambiente de experimentação foi utilizado um servidor com dois processadores Intel(R) Xeon(R) CPU E5-2630 v2, totalizando 12 núcleos físicos e 24 *threads* e 64 GB de RAM. No servidor foram criadas máquinas virtuais, conectadas a uma rede local estabelecida através de Linux Bridge. Neste ambiente foram criadas 17 máquinas virtuais, sendo 16 delas identificadas como VM *Workers*, com 1 vCPU e 1 GB de RAM, e uma VM *Manager*, com 2 vCPUS e 4 GB de RAM. Para garantir precisão nas medições de latência entre envios de mensagens, os relógios internos das VMs foram sincronizados.

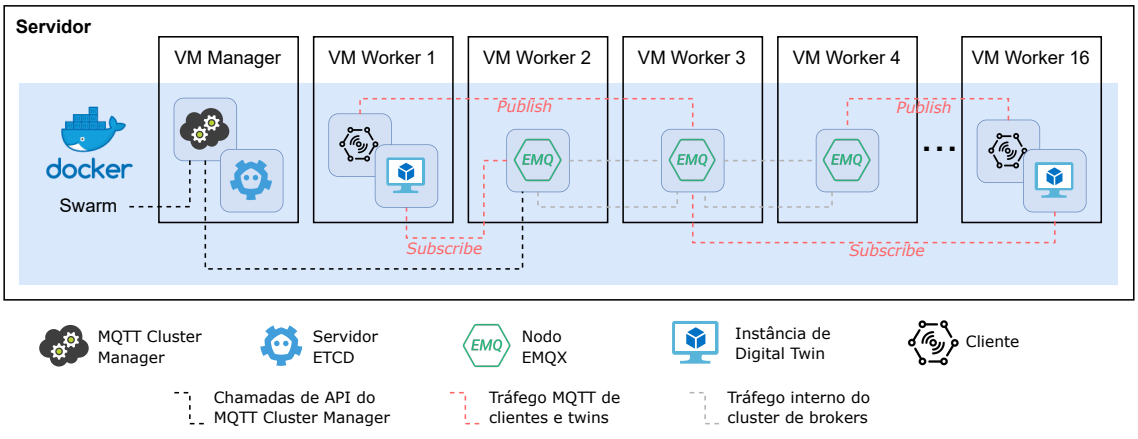


Figura 3. Implantação do experimento

A Figura 3 representa o ambiente de experimentação completo com todos os componentes utilizados para materializar o sistema de aquisição de dados para Gêmeos Digitais. Na *VM Manager*, ficam instanciados sempre os *containers* do *Cluster Manager* e o serviço do *etcd*, necessário para a composição do *cluster* de *brokers*. As VMs Worker recebem dinamicamente ao longo do experimento nodos do cluster instanciados através de réplicas de serviços do Swarm (controlados pelo Cluster Manager) e pares de clientes e instâncias de Gêmeo Digital. Os clientes fazem o papel de *publisher*, simulando o comportamento de cada uma das classes de aplicação definidas na Tabela 1. As instâncias simulam os subscribers contabilizando principalmente o tempo decorrido entre o envio (*publish*) da mensagem pelo cliente e o recebimento dela na instância. Vale ressaltar que os pares de clientes e instância de Gêmeo Digital são sempre alocados juntos em uma mesma VM para que os tempos de envio e recebimento de mensagens possam ser computados adequadamente.

Os seguintes testes foram realizados instanciando diferentes clientes simulando as categorias de serviços descritos na Tabela 1. Com base nos valores da tabela, foram configurados clientes V2X para enviar 3 200 000 mensagens, clientes simulando câmeras para enviar 750 000 mensagens, clientes simulando sensores de *Discrete Automation* mandando pacotes pequenos e grandes, ambos enviando 500 000 mensagens e um cliente crítico, para enviar 40 000 mensagens. Ao todo são enviadas 4 990 000 mensagens em cada rodada de experimento. Primeiramente foi avaliada a capacidade da arquitetura proposta de se replicar quando se baseando no tamanho da fila do *broker* (Subseção 5.2). Para isso foram realizados testes nos quais o *Cluster Manager* espera a fila atingir determinado tamanho, antes de adicionar um nodo novo ao *cluster*. Após isso o mesmo experimento foi repetido porém foram usadas métricas de mensagens em trânsito como parâmetro para justificar a expansão do *cluster* (Subseção 5.3). Em todos os experimentos foi considerado o número de mensagens perdidas em cada uma das categorias simuladas dando enfoque na perda e na latência de recebimento de mensagens críticas.

5.2. Escalando com o tamanho da fila

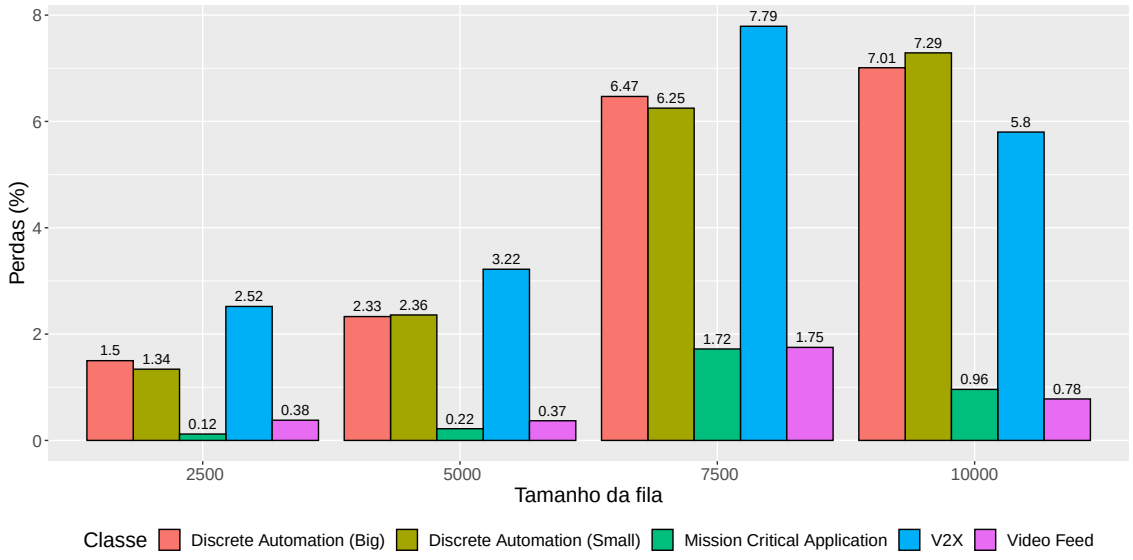
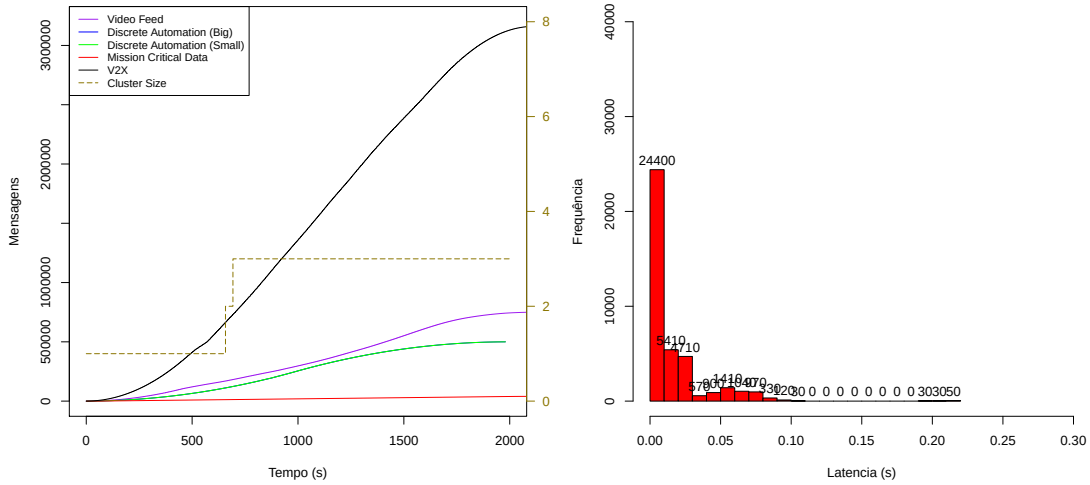


Figura 4. Experimentos no qual o *Manager* escala o *broker* de acordo com o tamanho da fila.

A partir de 10 execuções de experimentos utilizando o tamanho da fila como parâmetro para escalar o *cluster* foi gerado o gráfico da Figura 4 que apresenta as médias das perdas de mensagens separadas por classe de aplicação. Na configuração em que o *manager* aumenta o número de nodos no *cluster* quando o valor total da fila atinge 2500 mensagens, 1.967% das mensagens enviadas são perdidas, quando as configurações que utilizam 5000, 7500 e 10000 apresentam 2.599%, 6.553% e 5.284%. Entre os cenários considerados, aquele que possuiu a menor quantidade de perdas foi o que escala em 2500 mensagens. Analisando um dos cenários simulados com esta configuração, foi possível criar a Figura 5a, que mostra o números de mensagens recebidas ao longo do tempo neste experimento, além de em quais momentos do experimento mais nodos foram adicionados ao *cluster*. Foi gerado também o histograma da Figura 5b para representar as latências médias das mensagens críticas recebidas ao longo de todas as execuções, com o qual é possível verificar que para estes parâmetros, 97.30% das mensagens da classe de aplicações de missão crítica atendem os requisitos de tempo real especificados na Tabela 1, ou seja, foram entregues abaixo de 75ms.



(a) Mensagens recebidas **(b) Latência de Mensagens Críticas**

Figura 5. Mensagens recebidas e histograma da latência das mensagens no experimento que usa como parâmetro 2500 mensagens na fila.

5.3. Escalando com as mensagens em trânsito

Foi executada uma nova série de experimentos utilizando as mesmas configurações de clientes, porém, o *Cluster Manager* desta vez foi configurado para adicionar nodos ao *cluster* de acordo com o número de mensagens em trânsito. Utilizando essa métrica foi gerado o gráfico da Figura 6, onde foram testadas as configurações de 8, 16, 24 e 32[3] mensagens em trânsito. Para esses valores, as perdas de mensagens apresentadas foram em média 0.97%, 0.92%, 1.01%, 2.08% respectivamente. Analisando a configuração que usa 16 mensagens em trânsito é possível gerar a Figura 7a, que mostra as mensagens recebidas ao longo do tempo e momentos onde nodos foram adicionados ao *cluster*, e

[3]A saber, este valor (32) é o valor máximo configurado por padrão para mensagens em trânsito no broker EMQ X. Ao ultrapassar esse valor, um broker começa a reter mensagens e formar fila.

Figura 7b que apresenta um histograma das latências das mensagens críticas recebidas. Nesses gráficos é possível observar que os requisitos de tempo real da classe de aplicação de missão crítica foram atendidos para todas as mensagens.

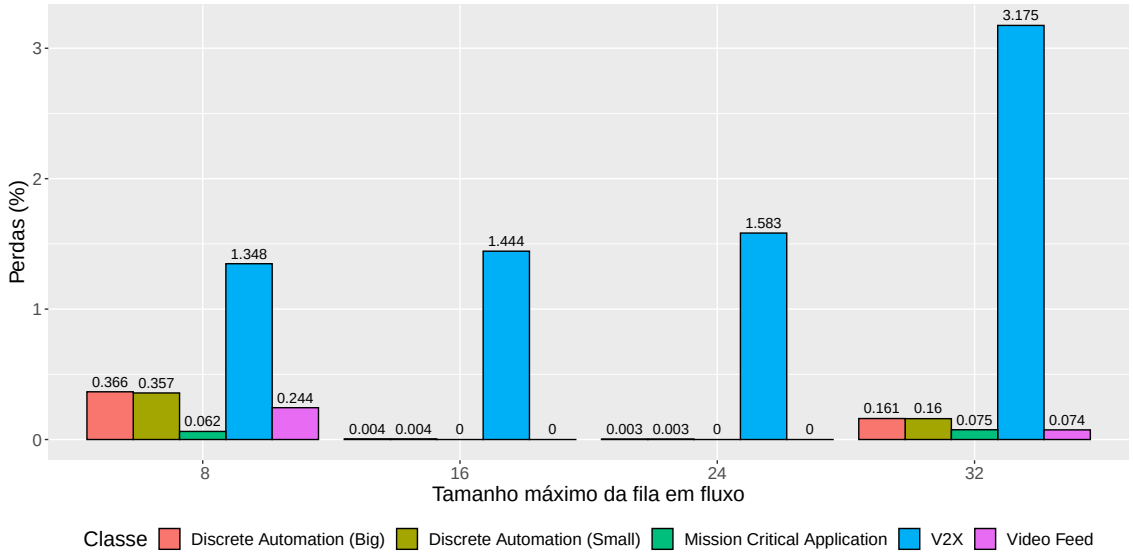
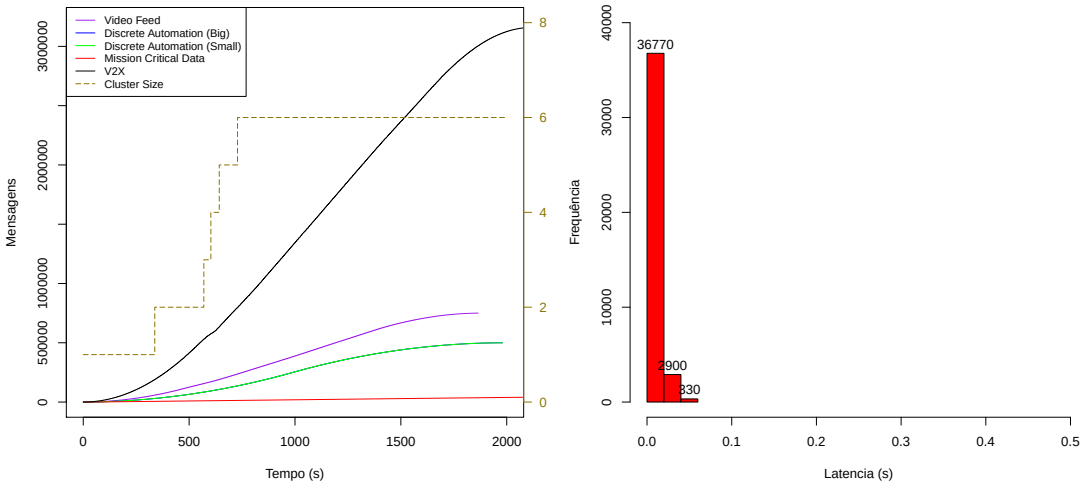


Figura 6. Experimentos no qual o *Manager* escala o *broker* de acordo com o número de mensagens em fluxo.



(a) Mensagens recebidas **(b) Latência das Mensagens Críticas**

Figura 7. Mensagens recebidas e histograma da latência das mensagens no experimento que usa como parâmetro 16 mensagens *inflight*.

Comparando esses experimentos, podemos constar que os experimentos que se baseiam nas mensagens em trânsito apresentam menos perdas do que os experimentos utilizando o tamanho da fila. Usar este parâmetro torna o *Cluster Manager* mais "sensível" sendo ele capaz de tomar decisões antecipando a formação de fila, porém isso pode fazer com que o *Cluster Manager* adicione nodos ao *cluster* sem que haja a necessidade,

gerando um desperdício de recursos. O experimento mostrado na Figura 5a, exibiu uma perda total de 0.92% das mensagens enviadas utilizando um *cluster* de 3 nodos, quando o da Figura 7a perdeu 0.80% usando um *cluster* de 6 nodos, ou seja, taxas de perda similares utilizando o dobro de recursos.

6. Conclusão

Neste trabalho, foi estudado o mecanismo de aquisição de dados para um Gêmeo Digital. Foram identificadas limitações como consequência do uso do protocolo MQTT quando o fluxo de dados é variável e muito grande. O uso do modo *cluster* nativo em algumas implementações de *broker* MQTT é capaz de amenizar esse problema, porém para isso é preciso dedicar recursos computacionais à mais uma instância de *broker*. Neste trabalho, foi proposto um componente adicional para o sistema de aquisição de dados para Gêmeos Digitais, ao qual chamamos de *Cluster Manager*, com o objetivo de escalar o *cluster* dinamicamente para que em situações onde o fluxo de mensagens varia. Este componente é capaz de evitar que recursos sejam desperdiçados quando a taxa de mensagens pode ser atendida por um número reduzido de *brokers* no *cluster* e evita também mensagens sejam perdidas ou sofram atrasos significativos quando essa taxa aumenta. Para validar o funcionamento do *Cluster Manager*, foram realizados uma série de experimentos onde clientes de diferentes classes foram simulados para enviar mensagens em taxas crescentes. Foram analisados aspectos dessa comunicação enquanto o *Cluster Manager* utilizou diferentes estratégias para escalar o *cluster* de *brokers*, se baseando em parâmetros como quantidade de mensagens em trânsito ou tamanho da fila. Foi concluído que a primeira apresenta em geral, menos perdas de mensagens pois como esse parâmetro é mais sensível e o *cluster* é escalado mais cedo. Também foi verificado que o uso de parâmetros muito sensíveis pode fazer com que o *Cluster Manager* inclua mais nodos no *cluster* do que necessário, gerando novamente desperdício de recursos.

Os experimentos que foram realizados não levaram em conta algumas configurações do EMQ X, como número máximo de mensagens em trânsito, que poderiam ser alteradas para produzir resultados diferentes. Neste trabalho, também não foi testado o uso do *Cluster Manager*, para diminuir a quantidade de nodos em um *cluster*, caso o fluxo esteja decrescendo. Para trabalhos futuros pretendemos realizar mais experimentos a fim de contemplar essas diferentes configurações possíveis, além de expandir o catálogo de serviços simulados. Fatores externos de rede como *jitter*, perda de pacotes e alta latência na rede não foram considerados durante a realização dos experimentos porém também são importantes para que seja possível simular com maior precisão situações mais realísticas.

Agradecimentos

Esse trabalho foi financiado em parte pelo Projeto PETWIN (financiamento FINEP e Consórcio de LIBRA), pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), pela Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) - Código de Financiamento 001.

Referências

3GPP (2021). Policy and charging control architecture. Technical Specification (TS) 23.203, 3rd Generation Partnership Project (3GPP). Version 17.2.0.

Banks, A., Briggs, E., Borgendale, K., and Gupta, R. (2019). MQTT Version 5.0. https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Online; acessado em 09 Ago, 2020.

Boschert, S. and Rosen, R. (2016). Digital twin—the simulation aspect. In *Mechatronic futures*, pages 59–74. Springer.

Cunha, B. and Batista, D. (2021). Avaliação da integração do protocolo mqtt em uma plataforma de cidades inteligentes. In *Anais Estendidos do XXXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 217–224, Porto Alegre, RS, Brasil. SBC.

Damjanovic-Behrendt, V. and Behrendt, W. (2019). An open source approach to the design and implementation of Digital Twins for Smart Manufacturing. *International Journal of Computer Integrated Manufacturing*, 32(4-5):366–384.

EMQ Inc. (2022). EMQ X. https://web.archive.org/web/20220120070016/https://www.emqx.io/. Online; acessado em 04 Fev, 2022.

Ferrari, P., Sisinni, E., Brandao, D., and Rocha, M. (2017). Evaluation of communication latency in industrial IoT applications. In *2017 IEEE International Workshop on Measurements & Networking (M&N)*, pages 1–6.

Grieves, M. (2016). Origins of the Digital Twin Concept. Technical report, Florida Institute of Technology / NASA.

Human, C., Basson, A. H., and Kruger, K. (2021). Digital Twin Data Pipeline Using MQTT in SLADTA. In Borangiu, T., Trentesaux, D., Leitão, P., Cardin, O., and Lamouri, S., editors, *Service Oriented, Holonic and Multi-Agent Manufacturing Systems for Industry of the Future*, pages 111–122, Cham. Springer International Publishing.

Koziolek, H., Grüner, S., and Rückert, J. (2020). A Comparison of MQTT Brokers for Distributed IoT Edge Computing. In Jansen, A., Malavolta, I., Muccini, H., Ozkaya, I., and Zimmermann, O., editors, *Software Architecture*, pages 352–368, Cham. Springer International Publishing.

Maaloul, S., Aniss, H., Kassab, M., and Berbineau, M. (2021). Classification of C-ITS Services in Vehicular Environments. *IEEE Access*, 9:117868–117879.

Mishra, B. (2018). Performance evaluation of MQTT broker servers. In *International Conference on Computational Science and Its Applications*, pages 599–609. Springer.

Thean, Z. Y., Voon Yap, V., and Teh, P. C. (2019). Container-based MQTT Broker Cluster for Edge Computing. In *2019 4th International Conference and Workshops on Recent Advances and Innovations in Engineering (ICRAIE)*, pages 1–6.

Trevisan, R., Knebel, F. P., and Wickboldt, J. A. (2020). Uma Avaliação do uso de MQTT para a Implementação de Digital Twins. In *Anais da XVIII Escola Regional de Redes de Computadores*, pages 41–47. SBC.