UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS–GRADUAÇÃO EM INFORMÁTICA APLICADA

VALDIR JOSÉ DA SILVA

AVALIAÇÃO DA SEGURANÇA DE BROKERS MQTT CONSIDERANDO ATAQUES DE NEGAÇÃO DE SERVIÇO

RECIFE – PE

2024

AVALIAÇÃO DA SEGURANÇA DE BROKERS MQTT CONSIDERANDO ATAQUES DE NEGAÇÃO DE SERVIÇO

Trabalho de Dissertação submetido à Universidade Federal Rural de Pernambuco, como requisito necessário para obtenção do grau de Mestre em Informática Aplicada sob a orientação do Prof. Dr. Fernando Antônio Aires Lins e coorientação do Prof. Dr. Milton Lima

RECIFE – PE

2024

FICHA CATALOGRÁFICA

Dados Internacionais de Catalogação na Publicação
Universidade Federal Rural de Pernambuco
Sistema Integrado de Bibliotecas
Gerada automaticamente, mediante os dados fornecidos pelo(a) autor(a)

S586a Silva, Valdir Silva
Avaliação da segurança de Brokers MQTT considerando ataques de negação de serviço / Valdir Silva Silva. - 2024.
84 f. : il.

Orientador: Fernando Antonio Aires Lins.
Coorientador: Milton Lima.
Inclui referências e apêndice(s).

Dissertação (Mestrado) - Universidade Federal Rural de Pernambuco, Programa de Pós-Graduação em Informática Aplicada, Recife, 2024.

1. Internet das coisas. 2. MQTT. 3. Denial of service. 4. Quality of service. I. Lins, Fernando Antonio Aires, orient. II. Lima, Milton, coorient. III. Título

CDD 004

UNIVERSIDADE FEDERAL RURAL DE PERNAMBUCO

VALDIR JOSÉ DA SILVA

Este Trabalho de Dissertação foi julgado adequado para a obtenção do título de Mestre em Informática Aplicada, sendo aprovado em sua forma final pela banca examinadora:

BANCA EXAMINADORA

Orientador: Prof. Dr. Fernando Antônio Aires Lins
Universidade Federal Rural de Pernambuco - UFRPE

Profª. Dra. Kádna Maria Alves Camboim Vale
Universidade Federal do Agreste de Pernambuco - UFAPE

Prof. Dr. *Obio*nor de Oliveira Nóbrega
Universidade Federal Rural de Pernambuco - UFRPE

Recife, 22 de Maio de 2024

Dedicatória

*A minha querida mãe, **Dona Santa**, por sempre acreditar em mim, muitas vezes sem entender os meus sonhos. No entanto, dedicou a sua vida em proveito das realizações e felicidade de seus filhos.*

*A Minha esposa, **Daiany Costa**, por sempre está ao meu lado de forma irrestrita e ao meu filho **Théo Vinícius**, o qual tive que renunciar muitas vezes a brincar com ele para realizar o mestrado.*

*Aos meus **irmãos**, que sempre estão por perto quando preciso.*

Agradecimento

Nestes anos de mestrado, de muito aprendizado, dedicação e empenho, também foram momentos de inúmeros desafios, perdas, dúvidas e alegrias. No entanto, tiveram pessoas especiais em cada um desses momentos. Portanto, expresso aqui, através de palavras, a importância delas no meu crescimento pessoal e profissional.

A Deus, pelo dom da vida e por me permitir realizar tão grandes sonhos. À minha família, principalmente na pessoa da minha esposa, que é meu porto seguro, meu expressivo agradecimento, por todas as vezes que precisei de carinho, compreensão e atenção. Agradeço aos demais familiares por sempre ter vocês ao meu lado, pois nunca foi fácil a nossa caminhada.

Ao Prof. Fernando Aires, pela orientação, dedicação, profissionalismo e por todo conhecimento compartilhado, tão importantes em minha vida acadêmica. Diversos momentos, em nossas reuniões online, cheguei desanimado e sem entender o que estava fazendo. No entanto, o senhor sempre com palavras de incentivo que resultavam em força e ânimo para continuar. Eu tenho a certeza de que sem o seu amparo, confiança e apreço nada disso seria possível. O senhor é mais do que um orientador, um verdadeiro amigo e mestre. Meu muito obrigado! Não poderia deixar de agradecer ao Prof. e coorientador Milton Lima pelas contribuições significativas em minha pesquisa.

Aos membros da banca examinadora, Profª Kádna Maria Alves Camboim Vale e Prof Obionor de Oliveira Nobrega, que tão gentilmente aceitaram participar e contribuir com esta dissertação. Aos professores das disciplinas eletivas, Prof. Marcelo Luiz, Prof. Gilberto Cysneiros e ao Prof. Tiago Ferreira, meu muito obrigado pelo conhecimento compartilhado em minha formação acadêmica.

Ao meu grande mestre, João Antônio, que mesmo sem o conhecer pessoalmente, me incentivou a ingressar no magistério com a sua didática maravilhosa. O seu conhecimento e dedicação profissional sempre foi um grande exemplo para mim. Meus sinceros agradecimentos.

Ao grande amigo, Félix Luís, um anjo que Deus colocou em meu caminho no momento da vida em que eu mais precisava. Obrigado pelos conselhos, disponibilidade e

ensinamentos, em momentos essenciais da minha vida. Sempre o considerei como um pai. Meu eterno agradecimento.

Aos meus amigos, Cléber Alberto, Thiago Valentim, Leonardo Lima, Diego Ribeiro, Jariedson Dantas, Paulo Martins e Victor Lira, que sempre me ajudaram quando tive dúvidas em relação ao conteúdo ou na caminhada do mestrado. Ou seja, vocês estiveram presentes no meu crescimento profissional. Meu muito obrigado pelos ensinamentos, companheirismo e principalmente pela amizade construída ao longo dessa jornada.

Por fim, a todos aqueles que contribuíram, direta ou indiretamente, para a realização desta dissertação, o meu sincero agradecimento.

Epígrafe

“Até você se tornar consciente, o inconsciente irá dirigir sua vida e você vai chamá-lo de destino. ”

(Carl Jung)

Resumo

A comunicação entre dispositivos IoT está se tornando cada vez mais comum em vários ambientes, incluindo hospitais, residências, agricultura, lavouras e outros. Entretanto os dispositivos IoT têm sido alvo cada vez mais comum pelos criminosos digitais. Portanto, estes objetos inteligentes, tornaram-se uma ameaça significativa à segurança dos sistemas baseados em IoT. Atualmente, o ataque de negação de serviço DoS *(Denial of Service)*, representa um dos ataques mais frequentemente realizados a sistemas IoT. Este ataque visa interromper parcialmente ou totalmente o fluxo total de comunicação entre dispositivos, aplicativos e redes. Este tipo de ataque possui diversas variações, como ataque de inundação, ataque de inundação baseado em *payload*, ataque de inundação baseado em QoS, entre outras variações. Estas variações impõem complexidade adicional na detecção e mitigação destes ataques. Assim, é importante que sejam realizadas configurações que mitiguem os impactos negativos causados por esses ataques. Esta dissertação, avalia o comportamento dos *Brokers* MQTT usados atualmente, considerando cenários de ataque de negação de serviço. Foram avaliados três *Brokers* MQTT: EMQX, Mosquitto e VerneMQ. A avaliação desses Brokers é realizada usando um cenário virtualizado por meio de métricas específicas para este contexto, como taxa de atendimento das requisições, taxa de erro, tempo de resposta, uso de CPU, memória e SWAP. Os ataques de negação de serviço aos quais os *Brokers* foram submetidos incluem: *Flooding connect*, *Flooding* baseado em *payload* e *Flooding* baseado em QoS. Os resultados mostraram que o *Broker* Mosquitto teve um desempenho melhor em relação as métricas de uso de CPU, memória e SWAP que os demais Brokers. Por sua vez, o EMQX teve um melhor desempenho em relação as métricas de atendimento das requisições e tempo de resposta no ataque de inundação CONNECT. Nos demais ataques, como os de *payload* e os de QoS, o Mosquitto teve um melhor desempenho em relação a essas métricas.

Palavras-chave: IoT, Internet das Coisas, MQTT, Denial of Service, Quality of Service, QoS.

Abstract

Communication between IoT devices is becoming increasingly common in various environments, including hospitals, homes, agriculture, crops and others. However, IoT devices have been an increasingly common target for digital criminals. Therefore, these smart objects have become a significant threat to the security of IoT-based systems. Currently, the DoS (*Denial of Service*) attack represents one of the most frequently carried out attacks on IoT systems. This attack aims to partially or completely disrupt the total flow of communication between devices, applications and networks. This type of attack has several variations, such as flooding attack, payload-based flooding attack, QoS-based flooding attack, among other variations. These variations impose additional complexity in detecting and mitigating these attacks. Therefore, it is important that configurations are made that mitigate the negative impacts caused by these attacks. This dissertation evaluates the behavior of currently used MQTT Brokers, considering *denial of service* attack scenarios. Three MQTT Brokers were evaluated: EMQX, Mosquitto and VerneMQ. The evaluation of these Brokers is carried out using a virtualized scenario through specific metrics for this context, such as request service rate, error rate, response time, CPU, Memory and SWAP usage. *Denial of service* attacks that Brokers were subjected to include: Connect Flooding, Payload-Based Flooding, and QoS-Based Flooding. Results showed that Broker Mosquitto performed better across CPU, memory, and SWAP usage metrics than other Brokers. In turn, EMQX had better performance in relation to request fulfillment and response time metrics in the CONNECT flood attack. In other attacks, such as payload and QoS attacks, Mosquitto had better performance in relation to these metrics.

Keywords: IoT, Internet of Things, MQTT, Denial of Service, Quality of Service, QoS.

.

Lista de Ilustrações

Figura 1 - Crescimento da internet das coisas 21
Figura 2 – Comunicação MQTT. 27
Figura 3 – Formato da mensagem MQTT. 29
Figura 4 – Comunicação utilizando QoS 0. 33
Figura 5 – Comunicação utilizando QoS 1. 34
Figura 6 – Comunicação utilizando QoS 2. 35
Figura 7 - Processo de avaliação de desempenho de sistemas segundo Raj Jain 43
Figura 8 – Cenário de experimentos. 50
Figura 9 – Taxa de atendimento de requisições com ataque *Flooding Connect* 57
Figura 10 – Tempo de resposta com ataque *Flooding Connect.* 58
Figura 11 – Utilização de CPU no ataque *Flooding Connect.* 59
Figura 12 – Utilização de memória RAM no ataque *Flooding Connect*. 60
Figura 13 – Taxa de atendimento de requisições com ataque *Flooding* baseado no *payload* de 5MB. 61
Figura 14 – Tempo de resposta do ataque *Flooding* com *payload* de 5MB. 62
Figura 15 – Utilização de CPU do ataque *Flooding* com *payload* de 5 MB. 63
Figura 16 – Utilização de memória RAM do ataque *Flooding* com *payload* de 5 MB 64
Figura 17 –Taxa de atendimento de requisições com ataque *Flooding* baseado no *payload* de 20 MB 65
Figura 18 – Tempo de resposta do ataque *Flooding* com *payload* de 20MB 66
Figura 19 – Utilização de CPU do ataque *Flooding* com *payload* de 20 MB 67
Figura 20 – Utilização de memória RAM do ataque *Flooding* com *payload* de 20 MB 67
Figura 21 – Taxa de atendimento de requisições do ataque *Flooding* com níveis distintos de QoS. 68
Figura 22 – Tempo de resposta do ataque *Flooding* com níveis distintos de QoS. 69
Figura 23 – Utilização de CPU do ataque *Flooding* com níveis distintos de QoS. 70
Figura 24– Utilização de memória RAM do ataque *Flooding* com níveis distintos de QoS. 70
Figura 25 – Informações adicionais sobre a avaliação do Broker Mosquitto 82
Figura 26 – Informações adicionais sobre a avaliação do Broker EMQX 83
Figura 27 - Informações adicionais sobre a avaliação do Broker VerneMQ 84

Lista de Tabelas

Tabela 1 – Visão comparativa dos trabalhos relacionados 41
Tabela 2 –. Características dos Brokers avaliados. 45
Tabela 3 – Carga de trabalho usada nos ataques 49
Tabela 4 - Quantidade de experimentos no ataque *Flooding connect*, *Flooding* baseado no *payload* de 5MB e 20MB 55
Tabela 5 - Quantidade de experimentos no ataque *Flooding* baseado em níveis de QoS 56
Tabela 6 – Resultados sobre a taxa de atendimentos das requisições para o ataque *Flooding Connect* 71
Tabela 7 - Resultados relacionados ao Tempo de Resposta para o ataque *Flooding Connect* 71
Tabela 8 -Resultados sobre a taxa de atendimentos das requisições para o ataque *Flooding payload* 72
Tabela 9 - Resultados relacionado ao tempo de resposta para o ataque *Flooding payload* 72
Tabela 10- Resultados sobre a taxa de atendimentos das requisições para o ataque *Flooding* QoS 73
Tabela 11- Resultados relacionado ao tempo de resposta para o ataque *Flooding* QoS 73

Lista de Abreviaturas

MQTT	*Message Queuing Telemetry Transport*
IoT	*Internet of Things*
DoS	*Denial-of-Service*
HTTP	*Hypertext Transfer Protocol*
COAP	*Constrained Application Protocol*
AMQP	*Advanced Message Queuing Protocol*
DDS	*Data Distribution Service*
XMPP	*Extensible Messaging and Presence Protocol*
CPU	*Central processing unit*
RAM	*Random access memory*
RFID	*Radio Frequency Identification*
TCP	*Transmission Control Protocol*
GPS	*Global Positioning System*
RF	*Radio Frequency*
UDP	*User Datagrama Protocol*
IP	*Internet Protocol*
QoS	*Quality of Service*
TLS	*Transport Layer Security*
IDS	*Intrusion Detection System*
DDOS	*Distributed Denial of Service*
API	*Application Programming Interface*
OWASP	*Open Worldwide Application Security Project*
WSN	*Wireless Sensor Network*

Sumário

1. Introdução 16
    1.1. Motivação 18
    1.2. Objetivos 19
        1.2.1. Objetivo geral 19
        1.2.2. Objetivos específicos 19
    1.3. Organização 19
2. Fundamentação Teórica 21
    2.1. Internet das Coisas 21
    2.1.1. Arquitetura de referência para IoT 22
    2.1.2. Segurança 24
    2.1.3. Segurança em IoT 24
    2.1.4. Negação de serviço (DoS) 25
    2.2. MQTT (Message Queuing Telemetry Transport) 27
    2.3. Transmissão MQTT 29
        2.3.1. Tipo de Mensagem 29
        2.3.2. DUP 31
        2.3.3. Nível de QoS 31
        2.3.4. Reter 31
        2.3.5. Comprimento do cabeçalho 32
        2.3.6. Comprimento do cabeçalho variável 32
        2.3.7. Carga útil 32
    2.4. Qualidade de Serviço (QoS) 32
        2.4.1 Considerações Finais 35
3. Trabalhos Relacionados 36
    3.1. Protocolo e Brokers MQTT 36
    3.2. Ataques de negação de serviço em Brokers MQTT 38
    3.3. Ataques de negação de serviço em divrersos protocolos 39
    3.4 Considerações finais 42
4. Avaliando a Resiliência de Brokers MQTT em Cenários de Ataques de Negação de Serviço 43
    4.1. Estabelecer objetivos e definir o sistema 44
    4.2. Listar serviços e possíveis resultados 47
    4.3. Selecionar métricas 47
    4.4. Listar parâmetros 49
    4.5. Selecionar fatores 50
    4.6. Selecionar a técnica de avaliação 51
    4.7. Selecionar carga de trabalho 51
    4.8. Planejar a execução dos experimentos 52
        4.8.1 Configurações e experimentos 52
        4.8.2 Ataques realizados 53
        4.8.3 Detalhamento do ambiente 54
        4.8.4 Realização dos experimentos 55

4.9. Analisar e interpretar os dados 57
4.9.1. Resultados do Ataque Flooding Connect 57
4.9.2. Resultados do Ataque Flooding baseado em payload 60
4.9.3. Resultados do Ataque Flooding baseado em QoS 68
4.10 Apresentar os resultados 71
5. Conclusões e Trabalhos Futuros 75
5.1. Conclusões 75
5.2. Contribuições 76
5.3. Limitações 76
5.4. Trabalhos Futuros 77
Referências 78
Apêndice A 82
Apêndice B 83
Apêndice C 84

1. Introdução

A Internet das Coisas (IoT) é uma área que está em significativa expansão, e que visa integrar software, hardware e objetos físicos através de redes de comunicação. A IoT fornece um ambiente que facilita a interação de diversos dispositivos em ambientes físicos e virtuais. Atualmente, bilhões de dispositivos estão conectados à Internet das Coisas, usando aplicações como automação residencial, sistemas educacionais, sistemas de saúde, entretenimento, agricultura e energia. Uma previsão sugere que aproximadamente 30 bilhões de dispositivos conectados à IoT estará em uso em 2030 [1].

Toda essa quantidade de dispositivos nos deixa propensos a muitos tipos de ataques. Um relatório de ameaças cibernéticas da SonicWall, do ano de 2023, registrou o segundo maior ano em registros e tentativas de ataques de *ransomware* pelo mundo, assim também como um grande aumento, em torno de 87% em malwares direcionados para a área de Internet das coisas. Outro destaque do estudo da SonicWall é o fato de que a América Latina observou um crescimento de 65% nos ataques focados em dispositivos IoT. Em todo o mundo, está marca chegou a 87% [2].

A crescente disseminação da IoT traz muitos benefícios, mas também aumenta a probabilidade de ataques e a vulnerabilidade às ameaças cibernéticas. Esses dispositivos existem em diversos aspectos do dia a dia das pessoas, como: coletando informações, lendo estas informações e armazenando-as. Nesse sentido, as soluções devem buscar mitigar os desafios de segurança e os problemas que os tornam propensos a diferentes tipos de ataques [3].

Neste contexto, os ataques de negação de serviço (DoS) têm implicações significativas quando direcionados à ambientes IoT, pois os mesmos geralmente têm recursos de hardware limitados, como memória, armazenamento, rede e energia [4]. Um dos problemas que esse tipo de ataque causa é a interrupção de serviços, comprometendo a capacidade de acessar dados e serviços em tempo real [5].

Um componente muito importante em ambientes MQTT são os Brokers [6], pois são eles que intermediam as mensagens entre os dispositivos. Ou seja, o Broker recebe a mensagem e as distribui aos assinantes que estão inscritos naquele tópico. Nesse sentido, é importante que se investigue a resiliência destes dispositivos a ataques de negação de serviço.

Uma avaliação em redes IoT de publicação e assinatura é essencial para garantir uma resposta rápida em ambientes que dependem de respostas em tempo real. Já que um atraso adicional pode afetar a disponibilidade do serviço de mensageria e causar retransmissões de mensagens [7].

Atualmente, em sistemas IoT, existem diversos protocolos, principalmente no contexto da camada de aplicação. Estes protocolos ditam as regras para a comunicação de dados entre os dispositivos, um exemplo destes protocolos inclue o HTTP como outros que vêm ganhando notoriedade no contexto da Internet das Coisas, como o *Constrained Application Protocol* (COAP) [8], o *Advanced Message Queuing Protocol* (AMQP) [9], o *Extensible Messaging and Presence Protocol* (XMPP) [10] e, por fim, o *Message Queue Telemetry Transport* (MQTT) [5]. O último citado, MQTT, é um dos protocolos mais adotados em ambientes IoT atualmente [11].

Os protocolos citados anteriormente têm características distintas em relação ao seu processo de interação, enquanto o COAP tem o estilo de *request-response,* o MQTT tem um tipo de interação publish-subscribe. Já o AMQP e XMPP suporta os dois estilos de interação. Destes protocolos, apenas dois trabalham com uma arquitetura, onde esteja presente a figura dos Brokers sendo eles: o AMQP e o MQTT [12]. A escolha pelos Brokers MQTT se deu por ele ser um protocolo muito utilizado atualmente e ser uma excelente escolha em cenários de IoT onde a comunicação é leve, a sobrecarga é mínima e o baixo consumo de energia são cruciais [13].

A principal contribuição desta dissertação é avaliar o comportamento de Brokers MQTT atualmente disponíveis no mercado, considerando ataques de negação de serviço. É muito importante fazer esse tipo de avaliação, devido ao aspecto crítico e sensível dos ambientes em que os serviços de mensagens são utilizados. Podemos citar, por exemplo, realizar experimentos nestes Brokers com carga intensa para ver como eles se comportam. Isso é crucial para saber se o sistema pode lidar com um grande número de dispositivos conectados e uma alta taxa de tráfego mantendo sua eficiência e resiliência.

Desta forma, a avaliação da resiliência em Brokers MQTT é fundamental para garantir a confiabilidade, a segurança e o desempenho contínuo em ambientes de comunicação críticos, onde a entrega de mensagens é fundamental para o funcionamento adequado do sistema [14].

1.1. Motivação

Em geral, sistemas e dispositivos não podem ser considerados 100% seguros, e sem qualquer tratamento de segurança podem ser explorados de inúmeras formas. Isto não é diferente quando falamos sobre sistemas e dispositivos IoT. Por exemplo, uma falha de segurança que usa o sistema multimídia de um automóvel para se conectar a outros softwares em um veículo pode ser usada para reprogramar estes softwares e, a partir desse momento, conseguir controlar todo o carro, como freios, direção, transmissão e motor [15].

Normalmente, existe uma grande preocupação relacionada a segurança da informação e ela está mais em foco quando os ataques acontecem em grandes corporações. Porém, grande parte dos problemas relacionados a diversos ataques também acontecem nas residências e equipamentos pessoais. Boa parte desses dispositivos, no universo IoT, funcionam com baixo consumo de energia ou sem sistema operacional e em sua maioria não podem processar soluções de segurança [16].

Isto também acontece com diversos serviços de mensagens que utiliza o protocolo MQTT; eles são muito utilizados, porém muitas vezes são mal configurados ou nem sequer são configurados. Uma simples atualização de serviço ajudaria a corrigir algumas falhas que o serviço poderia vir a ter. Este tipo de configuração evitaria alguns tipos de ataques. Diante de tudo isso, a cada ano tem-se um aumento significativo em relação aos ataques cibernéticos em sistemas IoT [2]. Analisar a resiliência desses dispositivos e destes serviços é importante para mitigar alguns dos principais ataques no ambiente de Internet das coisas.

Considerando que os serviços de mensageria MQTT são muito utilizados atualmente e produzem muitos dados, é muito importante avaliar a quantidade de carga e requisições que eles conseguem absorver sem causar a indisponibilidade do serviço.

Esta pesquisa foi motivada justamente por essa necessidade de comunicação entre os diversos dispositivos IoT e os Brokers MQTT. Pois, se estes ficarem indisponíveis, toda a comunicação do sistema IoT que utiliza o protocolo MQTT também ficará e isso causará a indisponibilidade do sistema (que é um requisito da segurança da informação). Perante o que foi apresentado, a pergunta que permeia essa pesquisa é: Qual o nível de resiliência em segurança atualmente oferecido por Brokers MQTT dentro de ambientes IoT?

1.2. Objetivos

A seguir são especificados os objetivos deste trabalho.

1.2.1. Objetivo geral

Esta dissertação tem como objetivo avaliar a resiliência da segurança de Brokers MQTT atualmente utilizados pela comunidade em cenários de ataques de negação de serviço.

1.2.2. Objetivos específicos

- Analisar a taxa de atendimento das requisições ao Broker, que é realizada de acordo com a quantidade de requisições enviadas ao Broker, e a quantidade que ele consegue encaminhar para os assinantes sem erros.

- Analisar o tempo de resposta dos Brokers avaliados, o qual podemos defini-lo da seguinte forma: como sendo o intervalo de tempo entre o início de uma requisição e a resposta dada pelo sistema [17];

- Analisar a porcentagem de CPU utilizada. Pode-se definir pela porcentagem de utilização no momento em que o ataque está acontecendo;

- Analisar a porcentagem de memória RAM utilizada. Pode-se definir pela porcentagem de utilização no momento em que o ataque está acontecendo;

- Analisar a porcentagem de SWAP utilizada. Pode-se definir que a mesma é utilizada quando toda a memória RAM do sistema já tenha sido consumida [18].

1.3. Organização

O restante desta dissertação está estruturado da forma que se segue:

O Capítulo 2 descreve os conceitos fundamentais para o entendimento desta pesquisa. Como por exemplo: internet das coisas e sua arquitetura, segurança, segurança em IoT,

negação de serviço, protocolo MQTT, transmissão e tipos de mensagem e por fim, qualidade de serviço.

O Capítulo 3 apresenta os trabalhos relacionados a esta pesquisa sobre resiliência em *Broker*s MQTT, como ataques de negação de serviço em ambientes IoT, assim como os principais desafios e a segurança que torna a internet das coisas propensa a esse tipo de ataque. Outro ponto importante também discutido nos artigos selecionados são os ataques DoS frente ao protocolo MQTT, assim como também a avaliação de desempenho de vários Brokers sobre as algumas métricas como CPU, memória RAM e SWAP.

No Capítulo 4 é apresentada a principal contribuição desta dissertação, que é a avaliação da resiliência de segurança de Brokers MQTT considerando ataques de negação de serviço. Foram avaliados Brokers atualmente usados pela comunidade, assim como três variações de ataques de inundação foram utilizados nessa avaliação e várias métricas dos subsistemas do hardware também foram avaliadas.

No Capítulo 5 são apresentadas as conclusões, as principais contribuições, limitações e os trabalhos futuros, que poderão ser realizados com base nesta dissertação.

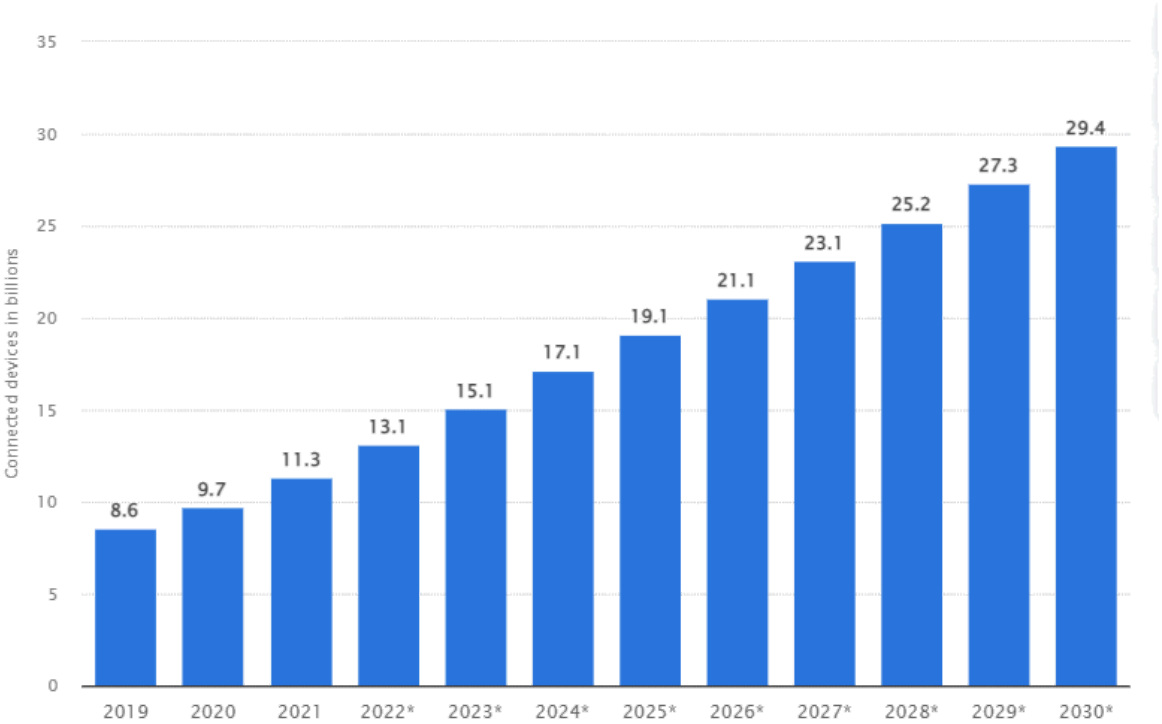
2. Fundamentação Teórica

Neste capítulo, são descritos conceitos fundamentais para o entendimento deste trabalho. Na Seção 2.1, é apresentado o conceito de Internet das Coisas e sua arquitetura. Assim como os conceitos de segurança, segurança em IoT e negação de serviço (DoS). A Seção 2.2 aborda o protocolo MQTT. Na Seção 2.3, é explorada a Transmissão e tipos de mensagens MQTT. A Seção 2.4 apresenta os níveis de qualidade de serviço existentes no protocolo MQTT. Por fim, a Seção 2.5 apresenta as considerações finais deste capítulo.

2.1. Internet das Coisas

O termo Internet das Coisas (IoT) ganhou muita popularidade na última década. Como mostra a Figura 1. A IoT se espalhou rapidamente para incluir aspectos de nossas vidas. Por exemplo, casas inteligentes, cidades inteligentes e dispositivos vestíveis de formas variadas [19]. Os dispositivos IoT trabalham para atingir diversos objetivos, ou seja, ela tem o potencial de transformar a vida das pessoas, proporcionando vários benefícios em diversas áreas. No âmbito da IoT, os serviços de mensageria desempenham um papel crucial, facilitando a comunicação entre os dispositivos de forma eficiente e assíncrona.

Figura 1: Crescimento da internet das coisas



Fonte: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

Os serviços de mensageria MQTT tem três componentes principais para que ocorra uma comunicação entre esses dispositivos, são eles: publicador, *Broker* e assinante. O *Broker* é o seu principal componente, ele é o responsável pela troca de dados entre os nós da rede MQTT, como sensores, atuadores, serviços ou mesmo outras plataformas. Eles são os responsáveis por publicar os dados dos nós na rede e disponibilizá-los para outros nós. Esse tipo de modelo de comunicação é conhecido como publicar/assinar, no qual sensores e atuadores atuam como publicadores e assinantes, respectivamente, e os usuários da aplicação podem atuar como publicadores ou assinantes dependendo da lógica de cada aplicação [20 - 21].

O protocolo utilizado por esses *Brokers* é o MQTT, o mesmo foi projetado para ser leve e eficiente em redes não confiáveis e de alta latência, tornando-o ideal para dispositivos IoT. No entanto, o protocolo MQTT é vulnerável a uma série de ameaças cibernéticas, uma das quais é a negação de serviço (DoS) [22]. Os ataques DoS evoluíram e se tornaram mais sofisticados devido ao aumento no número de dispositivos IoT incluídos [23]. As vulnerabilidades que podem ser exploradas para executar um ataque DoS no protocolo MQTT incluem tamanho de carga útil e níveis de QoS.

A segurança desses *Brokers* MQTT desempenha um papel crucial na preservação da integridade, confidencialidade e disponibilidade dos dados transmitidos em ambientes da Internet das Coisas (IoT). É muito importante destacar a necessidade urgente de avaliações abrangentes de resiliência neste tipo de serviço.

Nesta pesquisa apresentamos uma avaliação sobre a resiliência de *Brokers* MQTT considerando ataques de negação de serviço. Avaliamos três implementações difundidas de *Brokers* MQTT. Além disso, também avaliamos algumas métricas dos subsistemas de Hardware do *Broker* como CPU, memória RAM e SWAP, além do tempo de resposta e atendimento das requisições do serviço de mensageria em todos os *Brokers* avaliados.

2.1.1. Arquitetura de referência para IoT

Os sistemas IoT ainda não possuem camadas de arquitetura padrão, uma vez que se encontram pesquisadores que propõe uma arquitetura de três camadas como (Percepção/atuação, Rede e Aplicação) [24]; Outros que propõem quatro camadas como (Percepção, Rede, Middleware e Aplicação) [25]; e por fim, outros pesquisadores que propõem uma arquitetura de cinco camadas como (Percepção, Transporte, Processamento,

Aplicação e Negócio) [26]. Independentemente da arquitetura escolhida, cada uma dessas camadas tem problemas de segurança que as tornam propensas a diferentes tipos de ataques. Em nossa pesquisa a arquitetura de três camadas se adequa melhor pelas seguintes características: simplicidade e eficiência, escalabilidade moderada e por ser adequada para aplicações mais simples.

A seguir o detalhamento destas camadas:

- **Camada de percepção:** esta camada compreende os recursos físicos que fazem parte da IoT. Ela utiliza diversas tecnologias de sensores que geralmente são as "coisas da IoT" como (ex.: relógios inteligentes, sensores de temperatura, monitores cardíacos e máquinas de lavar) e os atuadores, além de ser um dispositivo para coletar dados, transforma-os em sinais digitais e os encaminha para a camada de rede [27]. As tecnologias da camada de percepção incluem tags RFID, câmeras, rede de sensores sem fio (WSN), GPS e Bluetooth. Esses dispositivos são escolhidos com base nas funcionalidades dos aplicativos IoT. As informações coletadas do ambiente podem vir de diferentes formas: movimento,luz, mudança de temperatura e localização [27].

- **Camada de rede:** Esta camada funciona exatamente da mesma forma que a camada de mesmo nome no modelo TCP/IP, ela desempenha um papel muito importante na comunicação entre os dispositivos que estão conectados ao ambiente da internet das coisas. É a camada responsável por transmitir as informações coletadas dos dispositivos da camada de percepção [28]. Esta camada armazena ou transmite os dados coletados para a camada de aplicação para processamento posterior. Quando se trata do contexto da internet das coisas, essa camada é a mais crucial, pois unifica as inúmeras formas de tecnologia de comunicação que possibilitam que dispositivos IoT se comuniquem entre si [29].

- **Camada de aplicação:** Quando se trata de interoperabilidade entre dispositivos IoT e suas redes de comunicação, a camada de aplicação é crucial. Esta camada que faz a mediação entre as atividades de um dispositivo IoT e a transferência de dados para a rede de forma que podemos utilizar. Diferentes considerações, incluindo o tipo de dispositivo e a tarefa a qual que ele executará, determinam qual o protocolo ideal para cada aplicativo de Internet das Coisas. São exemplos de protocolos dessa camada:

COAP, MQTT, AMQP, XMPP entre outros. Esta camada entrega os serviços aos usuários finais que os solicitaram [20]. Ela é a responsável pela lógica utilizada nos serviços IoT e fazer a interação desses serviços e aplicativos com os seus usuários [30].

2.1.2. Segurança

A segurança é um fator muito importante a se verificar ao longo da tomada de decisão de quais protocolos serão usados, porque alguns dos protocolos de comunicação, principalmente em ambientes IoT, não possuem mecanismos abrangentes de segurança. De acordo com a ISACA [31], a segurança da informação consiste em três princípios: confidencialidade dos dados, integridade dos dados e disponibilidade dos dados. Esta dissertação está ligada diretamente ao terceiro princípio, na medida em que a demora ou mesmo falta de resposta de um determinado *gateway* IoT impacta diretamente na disponibilidade do sistema como um todo.

Em relação a segurança em IoT, existem diversas considerações relevantes a serem feitas. Em primeiro lugar, a limitação do próprio dispositivo IoT (por exemplo, desempenho de computação e baixo consumo de energia) que pode requerer um protocolo de segurança mais leve e talvez com relativamente menor segurança [32]. Em segundo lugar, a heterogeneidade do ambiente é um desafio, pois cada dispositivo conectado pode usar diferentes protocolos e diferentes mecanismos de segurança. Por fim, a qualidade da rede que pode exigir uma adoção de mecanismos de segurança com impacto mínimo no desempenho [33]. Dado este contexto, a próxima seção explora a questão de segurança em IoT com mais profundidade.

2.1.3. Segurança em IoT

A crescente interconexão de dispositivos IoT introduz um conjunto significativo de desafios e ameaças. Em geral, esses dispositivos ou serviços possuem vulnerabilidades, que os invasores podem explorar para afetar a segurança destes dispositivos. Um dos ataques comuns e que atinge a segurança de ambientes baseados na internet das coisas é o ataque de negação de serviço. O principal objetivo dele é tornar os sistemas das vítimas inativos e inacessíveis para usuários legítimos por *malware* mal-intencionado [34]. Este tipo de ataque afeta um dos princípios básicos da segurança da informação, que é a disponibilidade.

A partir deste ponto, serão apresentados alguns tipos de ataques e as camadas IoT que os mesmos se relacionam.

A camada de percepção lida com a coleta de dados ainda em sua forma bruta, através de sensores e dispositivos. Um ataque bem conhecido nesta camada é o de interferência de rádio frequência (RF), onde o invasor intercepta e impede a comunicação entre os dispositivos IoT [25]. Outros exemplos de ataque incluem: espionagem, *spoofing*, ataques físicos, dentre outros.

A camada de Rede, por sua vez,tem um papel importante na comunicação entre os dispositivos, pois é a camada responsável por transmitir as informações coletadas dos dispositivos da camada de percepção [29]. Um tipo de ataque bem conhecido nesta camada é o flood UDP, onde o atacante envia um número alto de pacotes UDP *(User Datagrama Protocol)* em diferentes portas no alvo. Outro tipo de ataque comum nesta camada são os ataques de reflexão, ataques de modificação de dados, ataques de redirecionamento, dentre outros [30].

Por fim, a camada de aplicação entrega os serviços aos usuários finais que os solicitaram. Ela é a responsável pela lógica utilizada nos serviços IoT e faz a interação desses serviços e aplicativos com os seus usuários [30]. Esta camada sofre com diversos ataques também encontrados na camada de aplicação do modelo TCP/IP, como SQL *injection* e estouro de *buffer*.

2.1.4. Negação de serviço (DoS)

Um ataque de negação de serviço (DoS) é aquele em que o sistema ou rede de destino é intencionalmente sobrecarregado a ponto de não poder servir ao propósito pretendido. O objetivo do mesmo é tornar o sistema de destino inutilizável, sobrecarregando-o com tráfego ou fornecendo dados que causem falhas [35].

Quando um servidor é atingido por um grande número de solicitações ao mesmo tempo, diz-se que ele está passando por um ataque de inundação [36]. Já em relação a um ataque DoS contra o protocolo MQTT, o mesmo refere-se a uma grande quantidade de mensagens de publicação, assinatura e conexão que eventualmente esgotam os recursos do nó e tornam o nó incapaz de fornecer os serviços normalmente, causando assim a indisponibilidade.

Os ataques de negação de serviço relacionados ao protocolo MQTT têm muitos tipos e métodos distintos para deixar um sistema IoT indisponível.São eles: ataque de inundação connect, ataque de inundação baseado em carga útil, ataque de inundação baseado em qualidade de serviço (QoS), ataque de inundação de assinatura inválida, ataque de inundação TCP Syn entre outros [37]. Desta forma é essencial compreender cada tipo para mitigá-los e preveni-los.

Abaixo estão alguns tipos de ataque de negação de serviço sobre MQTT e suas respectivas descrições.

- Ataque de inundação Connect: é um ataque que visa sobrecarregar um *Broker* MQTT, abrindo múltiplas conexões usando o pacote CONNECT do referido protocolo.

- Ataque de inundação baseado em carga útil: é um ataque que tem como objetivo sobrecarregar um *Broker* MQTT, enviando mensagens com um volume de dados maior que o habitual para saturar os recursos do Broker.

- Ataque de inundação baseado em qualidade de serviço (QoS): é um tipo de ataque de negação de serviço (DoS) que explora os mecanismos de QoS implementados em redes para garantir a prioridade de certos tipos de tráfego do protocolo MQTT para sobrecarregar o *Broker*.

- Ataque de inundação de assinatura inválida: é um ataque malicioso, o qual um atacante tenta sobrecarregar um *Broker* MQTT enviando um grande volume de assinaturas inválidas ou malformadas. Este tipo de ataque pode ter como objetivo consumir os recursos do *Broker*, prejudicar sua capacidade de processar assinaturas legítimas ou até mesmo causar sua indisponibilidade.

- Ataque de inundação TCP Syn: é um tipo de ataque de negação de serviço (DoS) que explora a maneira como o protocolo TCP estabelece conexões. Este ataque visa esgotar os recursos de um servidor, tornando-o incapaz de processar solicitações legítimas. O mesmo pode ser direcionado a qualquer serviço baseado em TCP, incluindo o protocolo MQTT *(Message Queuing Telemetry Transport)*, que frequentemente utiliza TCP para comunicação.

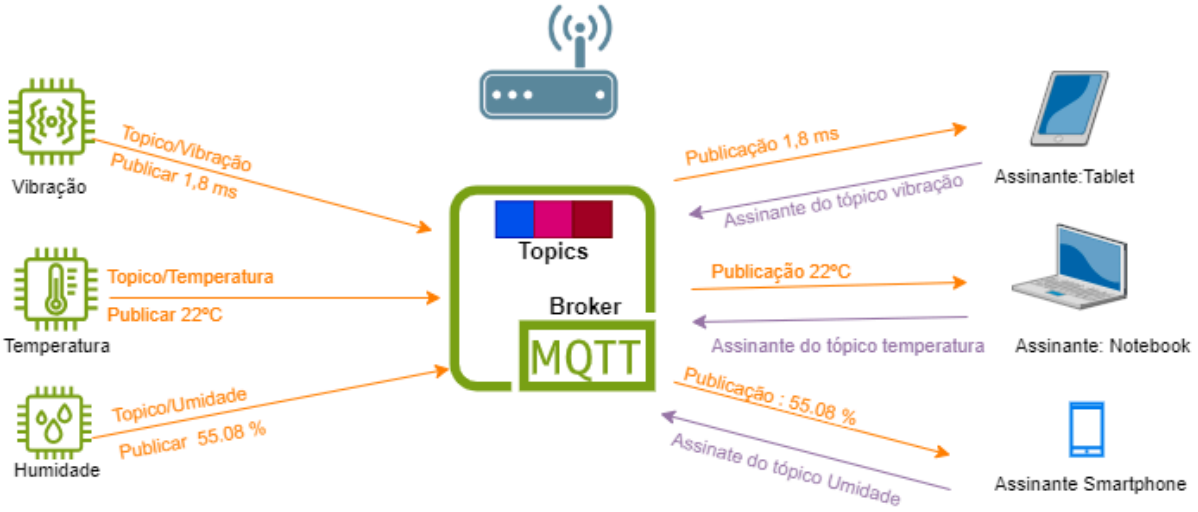
2.2. MQTT (*Message Queuing Telemetry Transport*)

O MQTT é um protocolo amplamente usado para comunicação de mensagens em ambientes IoT [38]. Ele é um protocolo de mensagens que usa o modelo de publicação/assinatura, ele foi originalmente projetado por Andy Stanford-Clark e Arlen Nipper. Atualmente, o MQTT é mantido pela OASIS Open (*Organization for the Advancement of Structured Information Standards)*.

O protocolo se popularizou pela simplicidade, baixo consumo de dados e pela possibilidade comunicação bilateral [39], Além de confiabilidade na entrega das mensagens e escalabilidade. Estas particularidades fazem do protocolo uma escolha popular para a implementação de comunicação eficiente e confiável em ambientes IoT, onde a interoperabilidade, a eficiência e a confiabilidade são aspectos críticos.

A Figura 2 mostra o uso do protocolo MQTT e de seus principais componentes que são: o publicador, o *Broker* e o assinante.

Figura 2: Comunicação MQTT



Fonte: Adaptado de (NERI; LOMBA, BULHÕES 2019)

O MQTT é um protocolo do tipo publicar/assinar e foi projetado para ser simples e leve, facilitando sua adoção em ambientes IoT. O MQTT é usado em vários cenários, como os setores:automotivo, industrial, telecomunicações e cidades inteligentes. Também suporta

comunicação assíncrona bidirecional um-para-muitos [40], escalabilidade para suportar muitos dispositivos e níveis básicos de QoS. Estes níveis são definidos como a confiabilidade e integridade da comunicação entre o publicador e o assinante. A qualidade de serviço possui três níveis distintos (nível 0, 1 e 2) que são aplicados no processo em duas etapas: enviar a mensagem do publicador para o *Broker* e transferir a mensagem do *Broker* para o assinante.

O Publicador MQTT é responsável por abrir e criar uma conexão, assim como fazer o envio de mensagens ao Broker. O publicador também é conhecido como editor. Já os assinantes se inscrevem em um tema de interesse com antecedência para receber essas mensagens enviadas. Os assinantes podem cancelar a assinatura de um tópico para parar de receber mensagens e fechar a conexão com o *Broker* [41][42]. Neste cenário, o *Broker* desempenha a função de servidor, coordenando as mensagens entre publicadores e assinantes.

Já o *Broker* é um componente central de um sistema de publicação/assinatura. As principais responsabilidades do *Broker* são: receber e filtrar mensagens, identificar quais assinantes estão inscritos em cada tópico e enviar mensagens para os tópicos corretos para que os assinantes as recebam. Em outras palavras, ele acompanha todos os assinantes. Finalmente, em uma comunicação MQTT, o publicador não se comunica diretamente com o assinante. Ele precisa do *Broker* para que essa comunicação ocorra [43].

Mesmo sendo tão utilizado, os sistemas baseados no protocolo MQTT podem apresentar falhas de segurança. A documentação do MQTT inclusive informa que é "responsabilidade do implementador fornecer recursos de segurança apropriados". Isto pode ser realizado usando o protocolo de segurança *Transport Layer Security* (TLS) [44]. Outras ações podem também ajudar na melhoria da segurança, como por exemplo: autenticação de usuários e dispositivos, autorização de acesso aos recursos do servidor, entre outras implementações. A mesma documentação também indica uma série de ameaças de segurança que os provedores de soluções devem considerar:

- Dispositivos podem estar comprometidos;
- Os dados inativos em clientes e servidores podem estar acessíveis;
- Comportamento do protocolo pode ter efeitos colaterais (ex: "*timing attacks*");
- Ataques de negação de serviço (DoS);
- As comunicações podem ser interceptadas, alteradas, reencaminhadas ou divulgadas;

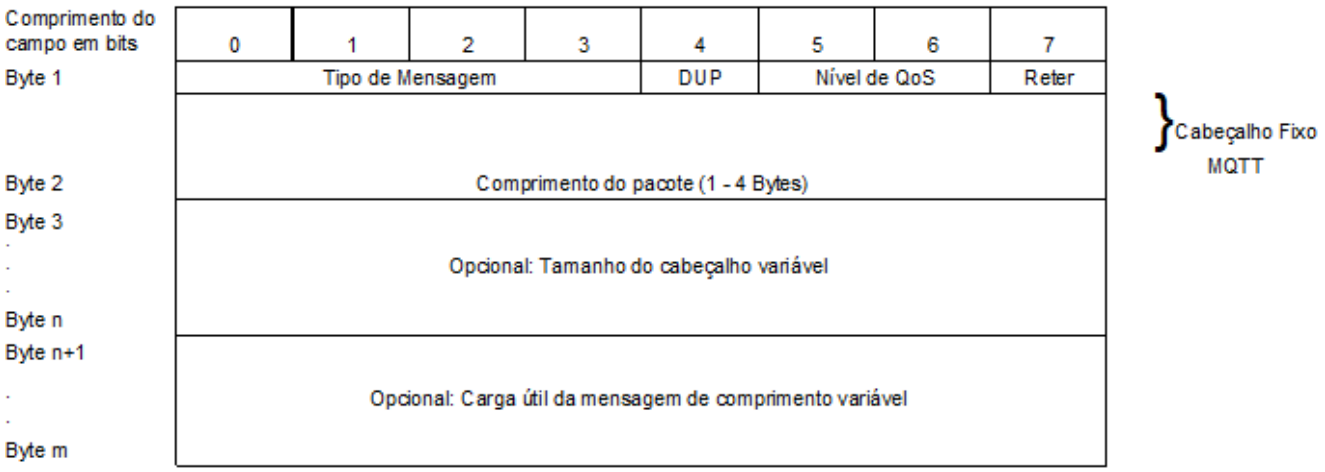
- Injeção de pacotes de controle falsificados.

2.3. Transmissão MQTT

O protocolo MQTT usa o protocolo TCP para transmissão de dados. O cabeçalho MQTT possui uma parte fixa que representa 2 bytes, então com a parte variável podendo atingir no máximo 5 bytes [43]. O primeiro byte é obrigatório, os quatro bits iniciais referem-se ao tipo da mensagem utilizada, o quinto bit indica se a mensagem é duplicada ou não, os próximos dois bits referem-se à qualidade de serviço, e o último bit indica se a mensagem deve ser retida ou não.

O restante dos bytes define o tamanho do pacote [40] - [42]. Essas informações podem ser visualizadas na Figura 3.

Figura 3 – Formato da Mensagem MQTT



Fonte: Adaptado de (RFWIRELESS, 2024).

2.3.1. Tipo de Mensagem

O primeiro campo do cabeçalho MQTT, se refere ao tipo de mensagem que podem ser encontradas em uma comunicação MQTT e como essas mensagens interagem com o publicador, *Broker* e assinante.

Abaixo estão os tipos de mensagens MQTT e suas respectivas descrições:

- **CONNECT:** Indica que a mensagem se destina a solicitação de conexão pelo publicador com o *Broker*. Ficando o publicador aguardando a resposta da conexão. Ou seja, essa mensagem inicia o processo de comunicação no protocolo MQTT;
- **DISCONNECT:** Indica que é uma mensagem enviada pelo assinante ao *Broker*. O notificando que não deseja mais se comunicar e está encerrando a conexão TCP/IP;
- **PUBLISH**: É um tipo de mensagem usada para publicar mensagens em tópicos ou receber mensagens publicadas por outros. A mensagem publicada está vinculada a um tópico ou assunto;
- **RESERVED**: Indica dois tipos de mensagens reservadas que estão proibidas de serem utilizadas;
- **CONNACK**: Indica que é uma mensagem de reconhecimento da conexão pelo *Broker*, onde o mesmo envia este tipo de mensagem CONNACK em resposta a uma solicitação CONNECT do publicador;
- **PUBACK:** Indica que é uma mensagem de reconhecimento da publicação, ou seja, é a resposta a um tipo de mensagem PUBLISH com QoS 1; este tipo de mensagem pode ser enviada pelo *Broker* em resposta a uma mensagem PUBLISH de um publicador ou por um assinante em resposta a uma mensagem PUBLISH do *Broker* como podemos ver na Figura 5;
- **PUBREC:** Indica que é uma mensagem de resposta a um tipo de mensagem PUBLISH com QoS 2; este tipo de mensagem pode ser enviada pelo *Broker* em resposta a uma mensagem PUBLISH de um publicador ou por um assinante em resposta a uma mensagem PUBLISH do *Broker* como podemos ver na Figura 6;
- **PUBREL:** Indica que é uma mensagem de resposta de um publicador a uma mensagem PUBREC do *Broker* ou do *Broker* para uma mensagem PUBREC de um assinante. Podemos ver na Figura 6, que ela é a terceira mensagem no fluxo de QoS;
- **PUBCOMP**: Indica que é uma mensagem de resposta de um *Broker*, a um tipo de mensagem PUBREL de um publicador ou a resposta de um assinante a uma mensagem PUBREL do *Broker.* É o último tipo de mensagem em fluxo de QoS como podemos ver na Figura 6;
- **SUBSCRIBE:** Indica que é uma mensagem de solicitação de assinatura para um ou mais nomes de tópicos;
- **SUBACK:** Indica que é uma mensagem de confirmação de assinatura, este tipo de mensagem é enviada pelo *Broker* aos assinantes confirmando o recebimento de uma mensagem SUBSCRIBE;

- **UNSUBSCRIBE:** Indica que é uma mensagem de solicitação, enviada pelo assinante ao *Broker* para cancelar a assinatura de um ou mais tópicos relacionado aquele assinante;
- **UNSUBACK:** Indica que é uma mensagem enviada pelo *Broker* para o assinante, reconhecendo o recebimento de uma mensagem UNSUBSCRIBE;
- **PINGREQ**: Indica que é uma mensagem de solicitação de ping enviada por um assinante conectado ao *Broker* para verificar se há conexão entre eles;
- **PINGRESP:** Indica que é uma mensagem de resposta enviada por um *Broker* a um PINGREQ mensagem.

2.3.2. DUP

Este é o segundo campo do cabeçalho, e ele indica se uma mensagem é duplicada, sendo reenviada devido ao fato que o destinatário não conhece a mensagem original. É usado para garantir uma maior confiabilidade de que a mensagem não foi duplicada – daí sua sigla *Duplicate Delivery.*

Caso aconteça algum problema com os ACK (*acknowledgment*), contudo, esta opção só se aplicará às mensagens cujo valor de QoS é diferente de 0 – caso contrário, "*fire and forget*" nesse nível não faria sentido [45]. Este campo é usado principalmente em situações onde a confirmação de entrega (QoS 1 ou QoS 2) está sendo usada.

2.3.3. Nível de QoS

Este campo melhora o nível de confiabilidade das mensagens no protocolo MQTT. A qualidade de serviço, nada mais é que um acordo entre o publicador e o *Broker*. Uma melhor confiabilidade da entrega de mensagens pode ser alcançada por meio de níveis mais altos de QoS; contudo, devido a problemas de latência, isso irá causar um atraso ou adicionará largura de banda de rede extra. Falamos brevemente do QoS nesta subseção, já que na próxima seção detalharemos melhor a forma de comunicação dos três níveis de qualidade de serviço no MQTT. As Figuras 4 a 6 mostram com mais detalhes esses níveis de QoS.

2.3.4. Reter

Este campo indica que a mensagem foi retida, a mesma será armazenada no Broker mesmo após despachá-la para todos os assinantes atuais. Se o mesmo tópico no Broker

receber uma nova assinatura, o novo assinante receberá as mensagens que estavam retidas. Essas mensagens no MQTT são pacotes PUBLISH com o sinalizador RETAIN definido como true (retain flag = 1). O MQTT possui várias propriedades e sinalizadores; uma dessas propriedades pode ser sinalizada com a flag RETAIN, que informa ao Broker que a mensagem deve ser armazenada [46].

2.3.5. Comprimento do cabeçalho

Este campo indica o comprimento restante do cabeçalho variável e dos dados de carga útil. Quando os pacotes são pequenos, com menos de 127 bytes, o comprimento restante é armazenado em um byte; o oitavo bit é um bit de continuação. Já para pacotes maiores que 127 bytes e menores que 16.383 bytes são usados 2 bytes [47]. É importante ressaltar que o tamanho máximo de um pacote MQTT é 256MB.

2.3.6. Comprimento do cabeçalho variável

Este campo indica o conteúdo do cabeçalho que é variável no MQTT, ele é usado apenas para fornecer informações adicionais quanto ao tamanho do cabeçalho variável. Ou seja, este campo não está presente em todos pacotes do protocolo. Por exemplo, no pacote CONNECT, este campo está presente e inclui o nome do protocolo, o nível de QoS, sinalizadores de conexão e o *Keep Alive*. Já o cabeçalho variável de um pacote PUBLISH inclui o nome do tópico e o identificador do pacote, caso o QoS não seja o nível 0 [48].

2.3.7. Carga útil

Por fim, existe o campo *payload*, também conhecido como carga útil. Ele é usado para atingir o objetivo principal do pacote. Assim como no protocolo TCP, esse campo representa os dados que estão sendo enviados. O pacote SUBSCRIBE funciona da mesma forma. O *payload* contém os tópicos a serem assinados, que é a tarefa principal deste tipo de pacote.

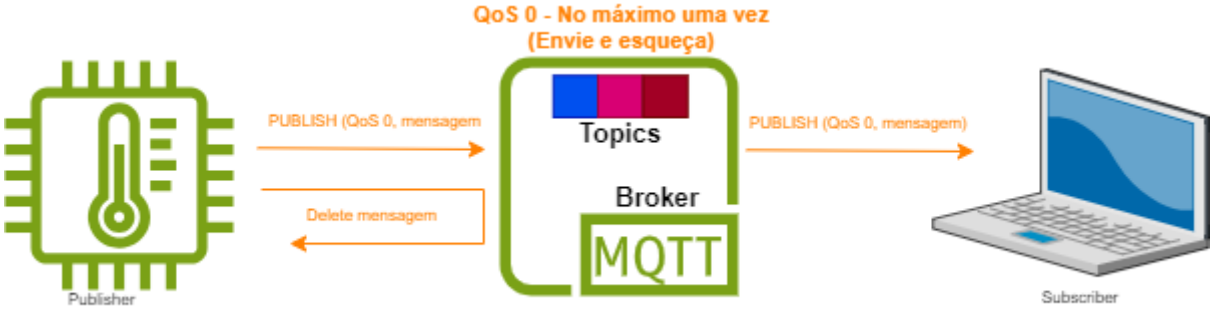
2.4. Qualidade de Serviço (QoS)

Esta seção detalha aspectos da Qualidade de Serviço (QoS) no MQTT, e que serão importantes para o entendimento da avaliação dos Brokers que será realizada posteriormente neste documento.

A qualidade de serviço encarrega-se de um papel muito importante em avaliações de resiliência, sobretudo onde a entrega confiável de mensagens é fundamental. Ela controla o nível de confiabilidade na entrega das mensagens entre publicador, *Broker* e assinante, esta confiabilidade tem consequências diretas na resiliência dos sistemas IoT [49]. Assim também como na recuperação de falhas, pois se houver alguma falha temporária, o uso do QoS pode ajudar, permitindo que as mensagens sejam enviadas até serem entregues com sucesso. Outro ponto crucial é o fluxo de controle, já que a escolha do nível de QoS afetará diretamente o *Broker* em relação a quantidade de mensagens que o mesmo terá que responder ao Publicador e ao assinante. Ou seja, quanto maior o nível de QoS mais mensagens serão enviadas ao Broker e isso o impacta diretamente na avaliação de desempenho.

Em relação aos níveis de QoS (qualidade de serviço) do protocolo MQTT, o padrão das mensagens é o nível 0, onde o publicador não armazena a mensagem e o destinatário não confirma o recebimento como mostra a Figura 4. Este método requer apenas uma mensagem PUBLISH e a mesma não é confiável, já que não existe uma confirmação por parte do destinatário [43].

Figura 4 – Comunicação utilizando QoS 0



Fonte: Adaptado de (MISHRA; MISHRA; Kertesz, 2021)

Além do nível padrão, temos outros dois níveis. O nível 1 funciona da seguinte forma. O publicador envia a mensagem **PUBLISH QoS 1** e aguarda a confirmação. Caso não receba esta confirmação, uma nova mensagem será enviada; neste nível de QoS a entrega é garantida, já que pelo menos duas mensagens são trocadas, como ilustra a Figura 5. Nesse modo de transferência, a mensagem é entregue pelo menos uma vez, havendo uma espera da recepção de *feedback* da entrega da mensagem, o chamado **PUBACK**. Não recebendo o **PUBACK**, a mensagem continuará sendo enviada até que haja o *feedback*.

No QoS 1, pode acontecer de a mensagem ser enviada diversas vezes e ser processada diversas vezes [50]. É importante observar que na QoS 1, se o cliente de publicação enviar a mesma mensagem novamente, ele definirá um sinalizador duplicado **(DUP).** Porém, este sinalizador é utilizado para fins internos e não é processado pelo Broker ou cliente. Independentemente do sinalizador DUP, o destinatário ainda envia um pacote **PUBACK** para confirmar o recebimento da mensagem, garantindo que o remetente esteja ciente do sucesso da entrega [51].

Figura 5 - Comunicação utilizando QoS 1

Fonte: Adaptado de (MISHRA; MISHRA; Kertesz, 2021).

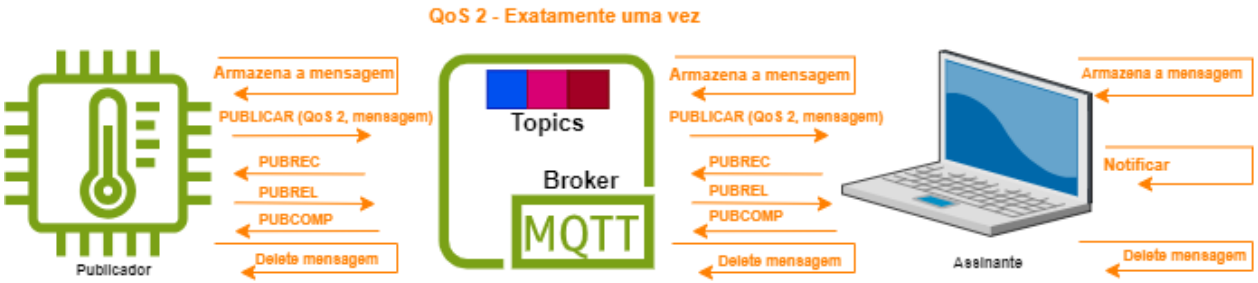
O último dos níveis garante a entrega de uma mensagem exatamente uma vez. Este método é o mais demorado em comparação com os anteriores, necessitando de quatro mensagens [43]. O publicador envia uma mensagem **PUBLISH QoS 2** e aguarda a confirmação. Quando o receptor recebe um pacote PUBLISH QoS 2 do remetente, ele processa a mensagem de publicação e responde ao remetente com um pacote **PUBREC** que reconhece o pacote **PUBLISH**. Ou seja, quem recebe envia a confirmação.

Se o remetente não receber a confirmação, ele enviará a mensagem novamente com um sinalizador **DUP** habilitado. Quando o remetente receber a confirmação, ele descarta a mensagem inicial. O remetente armazena o pacote **PUBREC** do receptor e responde com um pacote **PUBREL**, e o receptor descarta todos os estados armazenados e responde com um pacote **PUBCOMP** [51].

Quando o fluxo de QoS nível 2 for concluído, ambas as partes terão certeza de que a mensagem foi entregue e o remetente terá a confirmação da entrega. A Figura 6 detalha uma

comunicação MQTT utilizando o nível QoS 2.

Figura 6 - Comunicação utilizando QoS 2



Fonte: Adaptado de (MISHRA; MISHRA; Kertesz, 2021).

Como pode ser observado, o protocolo MQTT utilizando o nível de QoS 2 pode acabar utilizando muita memória RAM, processador e quantidade de dados transferidos. É exatamente isso que os criminosos digitais exploram para causar instabilidade ou indisponibilidade nas redes que usam esse protocolo. Um tipo muito comum de ataque que inunda a rede é o envio massivo de mensagens do tipo PUBLISH QoS 2 [52].

2.4.1 Considerações Finais

Neste capítulo foram descritos os principais conceitos teóricos que serão importantes para o entendimento deste trabalho e de suas contribuições. No próximo capítulo, serão apontados e detalhados trabalhos que estão relacionados à temática abordada nesta dissertação.

3. Trabalhos Relacionados

Este capítulo se destina a apresentar e discutir os trabalhos relacionados a temática desta dissertação, que é a avaliação da resiliência em segurança de Brokers MQTT considerando ataques de negação de serviço. A literatura atual acerca dessa temática, examina variadas perspectivas relacionadas à comunicação do protocolo MQTT, bem como alguns de seus elementos fundamentais, como publicadores, *Brokers* e assinantes. Na Seção 3.1 são apresentados alguns trabalhos que tem como tema central o protocolo MQTT e os Brokers MQTT. A Seção 3.2 o foco são trabalhos que estão relacionados a ataques de negação de serviço em Brokers MQTT. Já a seção 3.3 o objetivo dos trabalhos são em ataques de negação de serviço em diversos protocolos da camada de aplicação do modelo TCP/IP. Por fim, a Seção 3.4 apresenta as considerações finais deste capítulo.

3.1. Protocolo e Brokers MQTT

Bertrand-Martinez et al. [49], em 2020, propuseram uma metodologia de avaliação para Brokers IoT em abordagens quantitativas e qualitativas para classificação e avaliação desses serviços. Eles realizaram uma avaliação qualitativa usando o conjunto de padrões ISO/IEC 25.000 (SQuaRE) que é composta pelas seguintes etapas: estabelecimento de requisitos, especificação da avaliação, desenho da avaliação e execução da avaliação. Já o processo de Jain [17] para avaliação de desempenho foi composto por dez passos. Foram construídas métricas de recursos como: métricas de avaliação do gerente da rede e métricas de avaliação técnica. Assim como a construção de métricas de desempenho como: métricas de eficiência, confiabilidade e disponibilidade. Os autores validaram a viabilidade de sua abordagem metodológica com um estudo de caso em doze diferentes *Brokers* de código aberto. Esta pesquisa avalia algumas métricas quantitativas no momento em que os Brokers MQTT estão sobre stress, ou seja, quando está sob um ataque de negação de serviço DoS.

Koziolek et al. [53] compararam três Brokers MQTT em termos de escalabilidade, desempenho, extensibilidade, resiliência, usabilidade e segurança, em seu *gateway* de borda. O cenário de teste baseado em *cluster*, mostrou que o EMQX teve o melhor desempenho. Enquanto o HiveMQ não apresentou nenhuma perda de mensagem, já o Broker VerneMQ conseguiu entregar até 10.000 msg/s, respectivamente. Os autores também propuseram seis

pontos de decisão a serem levados em consideração pelos arquitetos de software para implantação de *Brokers* MQTT. Estão entre alguns pontos de decisão: configuração de segurança, configuração de carga de trabalho esperada, a seleção do *Broker*, se o mesmo deve ser de código aberto ou comercial entre outros.

Biswajeeban Mishra et al. [54] analisam e comparam o desempenho de implementações de Brokers não escaláveis como: o Mosquitto e o Bevywise MQTT e os *Brokers* escaláveis como: ActiveMQ, HiveMQ, VerneMQ e EMQX. Essa análise e comparação é feita utilizando CPU de núcleo único e as CPU multi-core, sob condições de estresse. Algumas métricas também são analisadas, como taxa máxima de mensagens, que é o número de mensagens enviadas por segundo, uso médio da CPU e latência média. Os autores chegaram à conclusão que o Mosquitto e Bevywise, tiveram um desempenho melhor do que outros *Brokers* escaláveis em um ambiente com recursos limitados. Já em um ambiente distribuído/multi-core, o ActiveMQ teve o melhor desempenho. Ele escalou bem e mostrou melhores resultados do que todos os outros corretores escaláveis. Contudo, o objetivo não foi analisar esses Brokers quando estavam sendo submetidos a ataques de negação de serviço. O principal ponto aqui que diferencia nossa pesquisa é que avaliamos a resiliência dos *Brokers* quando submetidos a ataques de negação de serviço, este é o foco principal desta dissertação.

Sicari et al. [55] apresentam um Sistema de Detecção de Intrusão (IDS) denominado REATO, que foi projetado para identificar ataques de negação de serviço (DoS) em Middlewares IoT. Um protótipo foi construído para validar a metodologia proposta. A motivação por trás desta pesquisa decorre da necessidade de conceber uma solução capaz de proteger objetos inteligentes contra-ataques de negação de serviço em ambientes IoT. O ataque DoS se baseia na geração de carga em demasia para a CPU do dispositivo, resultando em uma indisponibilidade temporária do serviço. REATO foi implantado em um Raspberry Pi, e buscou dar suporte a realização de análises de dados em tempo real, detecções e implementação de medidas de bloqueio.

Harsha et al. [60] usaram um mecanismo de busca chamado Shodan [61] e algumas APIs python para identificar várias brechas de segurança no protocolo MQTT. Os pesquisadores descobriram que a maioria dos servidores que hospedam o Broker MQTT não possui o mecanismo de autenticação e que podem ser explorados para roubar dados confidenciais. O artigo informa que foi implementado uma configuração experimental em um Raspberry Pi como um Broker MQTT e mecanismos de segurança foram implementados

editando alguns arquivos de configuração. Além disso, a Lista de Controle de Acesso (ACL) foi criada para garantir que os usuários tenham acesso apenas aos dados aos quais têm direito. Porém, uma lacuna encontrada é que o ataque de negação de serviço no protocolo MQTT não foi explorado.

Por fim, Sochor et al. [62] buscou detectar vulnerabilidades desconhecidas em diferentes implementações de *Brokers* MQTT. Os autores propuseram um método para gerar aleatoriamente sequências de teste (Randoop) para testar os diferentes *Brokers*, e eles conseguiram encontrar diversas falhas e exceções não tratadas. Foram implementados mais de vinte ataques em quatro *Brokers* distintos. Esta pesquisa adotou uma abordagem diferente, onde se testou diferentes ataques DoS em Brokers MQTT.

3.2. Ataques de negação de serviço em Brokers MQTT

Eric Gamess et al [56] avaliam como diferentes tecnologias de rede (Ethernet, WiFi nas bandas de 2,4 GHz e 5 GHz) e níveis de QoS podem produzir melhores resultados em diferentes tamanhos de payload. Eles consideram para seus experimentos um grande tamanho (variando de 512 a 1.048.576 bytes) para as mensagens publicadas através do MQTT. Eles também analisam o tempo de transmissão, o qual é explicado pelos pesquisadores que é o tempo que a mensagem sai do publicador, passa pelo Broker e chega até o assinante. Também é feita uma avaliação do comportamento do Mosquitto sob um ataque DoS, ao inundar o *Broker* com tráfego ilegítimo utilizando a ferramenta hping. Uma ferramenta própria de *benchmark* foi escrita na linguagem de programação C, com a biblioteca cliente MQTT do Mosquitto, usando MQTT v3.1.1. Foram realizados vários bancos de testes para realizar os experimentos em rede cabeada, redes Wifi, além das variações dos tempos de transmissão utilizando frequências distintas com a 2,4 GHZ e 5 GHZ. Por fim, foi avaliado o desempenho dos *Brokers* ActiveMQ, RabbitMQ e Mosquitto. Este trabalho relacionado tem pontos em comum com a pesquisa proposta na dissertação, mas existem algumas diferenças relevantes. Uma das discrepâncias notáveis reside no fato de que, na presente pesquisa, o ataque de negação de serviço foi conduzido em três Brokers. Além disso, foram analisadas métricas fundamentais, como a taxa de atendimento de requisições, tempo de resposta e as métricas relacionadas aos subsistemas de hardware, incluindo CPU, memória RAM e SWAP.

Hernández Ramos et al. [59] recomendou um modelo baseado em um *framework* de difusão para melhorar a segurança de aplicações e testou sua eficácia em duas

implementações MQTT. Usando um método *fuzzing,* os autores relataram ter verificado alguns problemas de segurança, uma ferramenta foi utilizada para testar os *Brokers* Moquette e Mosquitto e estes foram afetados por uma vulnerabilidade que possibilita um ataque DoS em configurações específicas. Essas vulnerabilidades descobertas testam a eficácia da ferramenta. Esta dissertação foca na execução de ataques de negação de serviço e alguns de seus tipos e a avaliação de desempenho de algumas métricas como: tempo de resposta e a taxa de atendimento de requisições, assim como algumas métricas relacionas aos subsistemas de hardware como processador e memória.

Por fim, Kotak et al. [42], em seu trabalho, buscou analisar *Brokers* disponíveis na Internet do ponto de vista da segurança. Para isto, foram realizados ataques de negação de serviço baseado no id da mensagem e no QoS padrão, além de técnicas de coleta de informações aos Brokers. Os autores compararam os resultados e se buscou evidenciar qual o *Broker* é o menos vulnerável. Os autores também objetivaram descobrir as vulnerabilidades de cada um dos *Brokers* testados. Após a execução dos ataques, foi realizada uma análise comparativa dos sete *Brokers* encontrados para os testes, a fim de observar as diferentes características que eles possuem quando alguns tipos de mensagens especiais são enviadas. O trabalho relacionado fez várias coisas, mas poderia também ter avaliado quando eles estavam sendo submetidos ao ataque de negação de serviço, utilizando os níveis de serviço disponíveis no MQTT, como o QoS 1 e 2, já que o nível padrão 0 foi testado.

3.3. Ataques de negação de serviço em divrersos protocolos

Tripathi e Hubballi [57] apresentaram uma pesquisa abrangente de ataques DoS na camada de aplicação e os classificaram, dependendo se são eficazes contra um protocolo específico ou contra um grande número de protocolos. Também foi discutido o funcionamento dos ataques e realizada uma comparação com base em diferentes parâmetros. Também descreveram ferramentas/utilitários/bibliotecas que podem ser usadas para lançar este tipo de ataque. Os autores resumiram fatos conhecidos sobre as características e mitigação de ataques DoS. Os autores também cobrem ataques de negação de serviço na Internet das coisas no âmbito teórico. Também foi feita uma comparação de vários produtos comerciais de mitigação de DoS com base em sua capacidade de combater diferentes ataques. Contudo, uma lacuna encontrada é que o ataque DoS especificamente no protocolo MQTT não foi explorado, assim como os experimentos não foram realizados.

Por sua vez, Alaa Alatram et al [58]. Descrevem um banco de testes IoT propositadamente desenvolvido para gerar um conjunto de dados chamado de DoS/DDoS-MQTT-IoT. A topologia para este banco de testes foi construída para facilitar o uso do protocolo MQTT e gerar um conjunto de dados IoT realista com foco em ataques DoS/DDoS. Este conjunto de dados, mostra variações de ataques de negação de serviço sobre o protocolo MQTT. O objetivo é ajudar a desenvolver e testar contramedidas contra tais ataques. Sendo assim, foi criado um ambiente de testes, com vários sensores físicos gerando dados e vários Raspberry Pis para comunicação entre publicadores, assinantes e *Brokers*. Outro ponto importante é que as máquinas invasoras foram usadas para criar dez cenários de ataque tanto para DoS quanto para DDoS contra o protocolo MQTT. Esse processo permitiu que dados de tráfego legítimos e de ataque fossem gerados e, portanto, coletados.

O objetivo foi analisar esses dados posteriormente e verificar se os ataques de negação de serviço estavam ocorrendo ou não contra o protocolo de mensagens. Esta pesquisa difere na medida em que avaliou o desempenho de três *Brokers* relacionando as seguintes métricas (tempo de resposta, taxa de atendimento das requisições, CPU, memória RAM e SWAP) contra-ataques de negação de serviço. Porém ainda faltou avaliar métricas como usabilidade, custo, conformidade entre outras.

A Tabela 1 apresenta os trabalhos citados nesta seção observando os seguintes critérios de comparação: I) Qual protocolo utilizado; II) se o objetivo é avaliar especificamente Brokers MQTT; III) se houve nos artigos uma avaliação sobre resiliência em segurança destes Brokers; IV) se existiu o foco em ataques de negação de serviço e quais os tipos desses ataques e V) quais ferramentas foram utilizadas para realizar o ataque.

O primeiro critério busca verificar qual protocolo utilizado, o próximo critério busca avaliar se os Brokers utilizam o protocolo MQTT. O terceiro critério tem como objetivo demonstrar se existe uma avaliação de resiliência em segurança relacionados a Brokers MQTT na literatura atual. O penúltimo critério de comparação busca verificar se os trabalhos realizaram ataques de negação de serviço e quais foram os tipos de ataques DoS realizados e por fim, o último critério busca verificar se foram utilizadas ferramentas e também agrupar estas ferramentas utilizadas.

Tabela 1 – Visão comparativa dos trabalhos relacionados.

Trabalhos Relacionados	Protocolo Utilizado	Foco em Brokers MQTT	Avaliação de resiliência	Tipos de Ataque DoS	Ferramenta de ataque
Bertrand-Martinez, Eddas, et al [49]	MQTT	Sim	Não	Não	Não
Koziolek H, Grüner S, et al [53]	MQTT	Sim	Sim	Não	Não
Biswajeeban Mishra et al [54]	MQTT	Sim	Não	Não	Não
Sicari et al. [55]	MQTT	Não	Não	Não	Não
Eric Gamess et al [56]	MQTT e TCP	Sim	Sim	*Payload*	Hping3
Tripathi e Hubballi [57]	HTTP/DNS/DHCP e NTP	Não	Não	Flooding/Slow DoS/ DoS NTP	Scapy/ Slowloris/Hulk /LOIC/DNS flood - png
Alaa Alatram et al [58]	MQTT	Não	Não	Sim	Hping3
Hernández Ramos et al. [59]	MQTT	Sim	Não	Não especificado	Scapy
Kotak et al. [42]	MQTT	Sim	Não	Baseado no id da mensagem e QoS 0	Não
Harsha et al. [60]	MQTT	Sim	Não	Não	Não
Sochor et al. [62]	MQTT	Sim	Não	Não	Não
Este Trabalho	MQTT	Sim	Sim	Flooding Connect/ *Payload*/QoS	JMeter

Fonte: O autor (2024).

Inicialmente, verificou-se que apenas um dos trabalhos não utilizou o protocolo MQTT, o que mostra a importância deste protocolo para o ambiente da Internet das coisas. Muitos artigos propuseram a utilização e avaliação de Brokers MQTT, porém apenas dois [53][56] realizaram algum tipo de avaliação com intuito de verificar a resiliência desses Brokers.

Alguns dos artigos apresentaram diversos ataques de negação de serviço, porém não realizaram os ataques em si. Um desses enfoques está centrado na negação de serviço. No entanto, foi conduzida uma avaliação utilizando um conjunto de dados [58], empregando uma

variedade de técnicas convencionais de aprendizado de máquina comumente utilizadas em sistemas de detecção de intrusão (IDS). O objetivo principal consistia no desenvolvimento e teste de contramedidas.

Dois artigos realizaram os experimentos o primeiro realizou ataques de negação de serviço baseado no aumento do *payload* [56], enquanto que o segundo realizou os ataques baseado no id da mensagem e no QoS padrão [42]. Apesar de especificar os ataques um deles [42] não evidencia qual ou quais ferramentas foram utilizadas nos experimentos para realizar os ataques. Apesar de uma das pesquisas ser muito semelhante à desta dissertação [56], temos algumas diferenças. A primeira delas é que o ataque de negação de serviço na pesquisa comparada foi realizado em apenas um *Broker*. Além disso, métricas importantes como taxa de atendimento de requisições e as métricas relacionadas aos subsistemas de hardware não foram avaliadas. Finalmente, o artigo realiza apenas um ataque de negação de serviço, que é o ataque baseado em *payload.* Esta dissertação realiza três tipos de ataques distintos, como será detalhado no próximo capítulo.

3.4. Considerações Finais

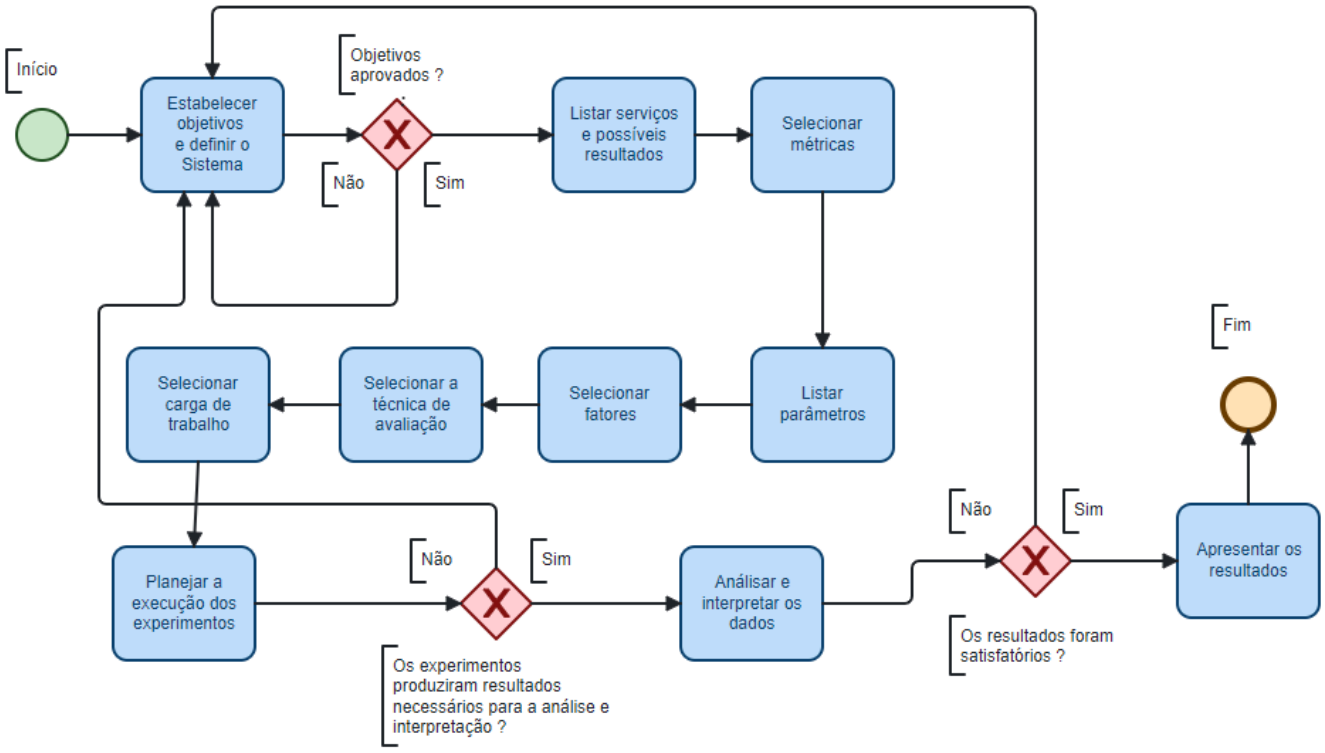
Este capítulo apresentou uma visão geral do estado da arte sobre a avaliação de resiliência de segurança de Brokers MQTT considerando ataques de negação de serviço. Algumas propostas até avaliam esses ataques; no entanto, poucos trabalhos se baseiam especificamente nos tipos de ataque DoS para MQTT. Outro ponto a ser observado é que as avaliações desenvolvidas não descrevem com a devida profundidade o processo de como foi realizado o ataque e quais ferramentas utilizadas.

Por fim, é visível o número reduzido de trabalhos que apresentam uma avaliação de resiliência com foco específico em ataques de negação de serviço em Brokers MQTT. Toda está problemática serve de motivação para a contribuição principal desta dissertação, que é avaliar o quão resiliente são os Brokers MQTT testados, considerando cenários de ataque de negação de serviço. A ideia é evidenciar os resultados desta avaliação para que se possa fazer uma avaliação de *Brokers* que estão sendo comumente utilizados atualmente pela comunidade. A contribuição secundária é avaliar métricas de desempenho e o comportamento do hardware desses *Brokers* quando estão sob ataque.

4. Avaliando a Resiliência de Brokers MQTT em Cenários de Ataques de Negação de Serviço

O objetivo principal desta dissertação é à avaliação da resiliência em segurança de Brokers MQTT frente a ataques de negação de serviço. Para atingir este objetivo, e para melhor estruturar a avaliação, uma referência clássica e amplamente adotada na área de avaliação de desempenho de sistemas é adotada, Jain [17]. É importante enfatizar que Jain é uma inspiração e não houve preocupação formal em seguir todos os passos que Jain propõe. Jain [17] propõe uma metodologia que prevê uma série de passos, bem definidos e sequenciais, para a avaliação de desempenho de um sistema. O processo proposto por Jain é dividido em 10 passos, como ilustra a Figura 7.

Figura 7 - Processo de avaliação de desempenho de sistemas proposto por Raj Jain



Fonte: Adaptado de (Bertrand-Martinez; Feio; Nascimento; Kon; Abelém, 2020)

Os passos são descritos da seguinte forma:

- 1- estabelecer objetivos e definir os limites do sistema que é o primeiro passo em qualquer projeto de avaliação de desempenho;
- 2 - listar os serviços e possíveis resultados, já que cada sistema possuí um conjunto de

serviços e ao solicitar qualquer um destes serviços, o usuário pode ter como resposta vários resultados possíveis;

- 3 - selecionar métricas, com o objetivo de comparar o desempenho, normalmente essas métricas estão relacionadas a disponibilidade, velocidade, precisão etc;
- 4- listar os parâmetros, ou seja criar uma lista de parâmetros que afetam o desempenho, por exemplo, parâmetros do sistema como hardware e software entre outros;
- 5 - selecionar os fatores, este passo está relacionado com o passo anterior, já que a lista de parâmetros pode ser dividida em duas partes, durante a avaliação. Os parâmetros que são variados e os que não são variados, os que são variados são chamados de fatores e seus valores são conhecidos como níveis;
- 6- selecionar a técnica de avalição, este passo têm três técnicas que são possíveis de se utilizar: modelagem analítica, simulação e medição;
- 7- selecionar carga de trabalho, este passo consiste em uma lista de solicitações feitas pelo serviço ao sistema;
- 8- Planejar a execução dos experimentos , este passo é necessário depois de definido uma lista de fatores e seus níveis. Essa sequência de experimentos deve proporcionar um maior volume de informação com o menor esforço possível;
- 9- Analisar e interpretar dados, este penúltimo passo é muito importante, pois permite identificar padrões, tendências e áreas de melhoria, fornecendo uma base sólida para tomada de decisões;
- 10- apresentar os resultados, a fase conclusiva de todos os projetos de desempenho envolve a comunicação dos resultados e que esses dados sejam apresentados de uma maneira compreensível.

A avaliação da resiliência de Brokers MQTT, proposta nesta dissertação, segue os passos de Jain. Estes passos se encontram detalhados nas subseções a seguir.

4.1. Estabelecer objetivos e definir o sistema

Em um projeto de avaliação, o primeiro passo é estabelecer os objetivos do estudo. Após definidos esses objetivos, segue-se a definição do sistema, que consiste em identificar o escopo do sistema a ser testado e delinear suas fronteiras. Os objetivos do estudo norteiam todo o projeto e execução da avaliação a ser realizada. Por sua vez, a definição das fronteiras

do sistema afeta diretamente, por exemplo, as métricas de desempenho a serem escolhidas. Esta definição permite também uma melhor compreensão do escopo da avaliação a ser realizada.

Na avaliação proposta nesta dissertação, o objetivo principal é avaliar a resiliência, em termos de segurança, relacionada aos principais Brokers MQTT usados atualmente em sistemas IoT. A pesquisa avaliou a resiliência de três Brokers MQTT diante de ataques de negação de serviço. Os tipos de ataques direcionados aos Brokers incluem: ataque de conexão por inundação (Flooding Connect), ataque de inundação com base em carga útil (*payload*), e ataque de inundação com base na qualidade de serviço (QoS).

A avaliação apresentada nesta dissertação será realizada em três Brokers MQTT. O critério para selecionar esses Brokers incluí ser de código aberto, ter uma comunidade ativa de usuários, suporte a versão 5 do MQTT e possuir pelo menos documentação básica. Inicialmente, tínhamos um escopo de cinco *Brokers* a serem avaliados, porém o *RabbitMQ* [75] não suportava a versão 5 do MQTT e o *NanoMQ* [76] por ser uma solução nova, ainda não tem uma comunidade com muitos usuários em relação aos Brokers escolhidos.A Tabela 2 apresenta as principais características dos Brokers avaliados.

Tabela 2 - Características dos Brokers avaliados

	Mosquitto	**VerneMQ**	**EMQX**
Linguagem	C	Erlang	Erlang
Versão do Broker	2.0.1	1.13	5.1.6
Níveis de qualidade de serviço	0, 1 e 2	0, 1 e 2	0, 1 e 2
Versão do protocolo	3.1, 3.1.1 e 5	3.1, 3.1.1 e 5	3.1, 3.1.1 e 5

Fonte: O Autor (2024).

Todos os Brokers avaliados implementam totalmente as versões 3 e 5 do protocolo MQTT e estão em sua versão mais recente. Além disso, eles suportam todos os níveis de QoS proposto pelo padrão MQTT que são os níveis (0,1 e 2). O Mosquitto é escrito em C, e os outros em Erlang. Erlang é uma linguagem de programação usada para construir sistemas soft real-time, ou seja, são sistemas computacionais que operam com restrições temporais, mas com uma tolerância aceitável para a conclusão de suas tarefas dentro de prazos específicos.

Além disso são sistemas massivamente escaláveis com requisitos de alta disponibilidade [63].

Abaixo, os Brokers escolhidos são detalhados:

- **EMQX** É um Broker MQTT com mecanismo de processamento de mensagens em tempo real. Ele implementa o protocolo MQTT em suas diversas versões, conforme mostrado na Tabela 2. Este Broker inclui suporte para intervalos de expiração de sessão e mensagem, propriedades de usuário, assinaturas compartilhadas e "apelidos" para tópicos, entre outros recursos. É um Broker escalável, ou seja, ele pode lidar com milhões de conexões MQTT simultâneas. O EMQX tem opções para uso na nuvem, localmente e por meio de contêineres [64].
- **VerneMQ** Assim como o EMQX, o VerneMQ é um serviço de publicação e assinatura, ou seja, ele é um *Broker* de mensagens que implementa o protocolo MQTT. O projeto teve início em 2014 e conta com características importantes, como suporte a persistência dos dados, escalabilidade e tolerância a falhas de mensagens MQTT. Ele usa tecnologia de *cluster* sem mestre. Sendo assim, os mestres e os escravos não têm função "especial", o que o torna mais simples e seguro. Ele pode ser modificado e reutilizado porque é de código aberto e licenciado sob Apache 2 [65]. O que difere a tecnologia de *cluster* sem mestre do VerneMQ do P2P (peer-to-peer) é que neste, cada nó é igual e possui as mesmas responsabilidades e capacidades. Enquanto no primeiro, todos os nós do *cluster* tenham funções semelhantes, há uma coordenação centralizada para gerenciar a distribuição de tópicos, partições e manutenção de estado distribuído. Isto o difere da natureza descentralizada de um sistema P2P.
- **Eclipse Mosquitto** O projeto Mosquitto foi inicialmente desenvolvido em 2009 e, ao contrário do EMQX e VerneMQ, é um Broker com uma arquitetura distinta, já que o mesmo é de *thread* única. Isso significa que todas as operações e tarefas dentro do programa são executadas sequencialmente dentro desse único *thread*, sem paralelismo. Ele implementa o protocolo MQTT em todas as suas versões. Seu *design* leve o torna apropriado para ser implantado em redes IoT, especialmente em dispositivos embarcados que possuem recursos computacionais limitados e são de baixo custo. [66]. Mosquitto é um dos Brokers de código aberto mais utilizados no mundo [67]. Este Broker alcançou esta posição com particularidades como: facilidade de configuração, leveza e uma comunidade ativa. Um dos seus problemas é que não possui suporte a *cluster*. Ou seja, não funciona como um conjunto de servidores, o que

o deixa com escalabilidade limitada, especialmente se compararmos com outros Brokers no mercado [68].

4.2. Listar serviços e possíveis resultados

O passo seguinte na análise do sistema é listar os serviços dos Brokers MQTT. Quando um usuário solicita qualquer um desses serviços, diversos resultados podem ocorrer. Por exemplo, em um sistema de mensageria, a publicação da mensagem pode chegar ao Broker ou ao assinante corretamente, incorretamente (devido a diversos fatores) ou mesmo não ser respondida devido a indisponibilidade do Broker

O sistema analisado (Broker MQTT) oferece alguns dos seguintes serviços como: publicação, assinatura, retenção e persistência de mensagens, gerenciamento de sessão, qualidade de serviço, integração com outros protocolos, entre outros. Para a nossa avaliação vamos focar especificamente nos três serviços abaixo.

- **Publicação e assinatura de mensagens:** permite a publicação de mensagens pelos publicadores e facilita a assinatura de tópicos por dispositivos assinantes;
- **Gerenciamento de tópicos**: é possível criar, modificar e excluir tópicos, assim como garantir a organização e hierarquia adequadas dos tópicos para facilitar a comunicação;
- **QoS (Quality of Service - Qualidade de Serviço)**: oferece suporte a diferentes níveis de QoS, possibilita a escolha do nível adequado, conforme a necessidade da aplicação.

É importante destacar que a quantidade destes serviços pode divergir nos Brokers MQTT disponíveis no mercado.

O sistema avaliado deve prover como resultado a publicação das mensagens chegando ao Broker com suas respectivas cargas e com seus níveis distintos de QoS. Considerando o sistema nesta condição de carga e qualidade de serviço, pretende-se obter o melhor ajuste dos fatores para a avaliação da resiliência do serviço de mensageria.

4.3. Selecionar métricas

Estabelecer as métricas do sistema a ser avaliado significa escolher os critérios para a verificação do desempenho do mesmo. Em geral, estas métricas podem ser bem variadas, como: velocidade, consumo, disponibilidade, dentre outras. A resiliência dos Brokers MQTT pode ser analisada considerando as seguintes métricas: tempo de resposta, taxa de atendimento das requisições, utilização de CPU, utilização de memória RAM e por fim, a utilização da memória de disco, conhecida como SWAP.

Como consequência da métrica, taxa de atendimento das requisições, também analisamos a taxa de erro, que é a porcentagem de requisições enviadas, mas que não chegaram aos Brokers avaliados.

- **Tempo de resposta** Este é o tempo que o Broker leva para processar a solicitação;
- **Taxa de atendimento das requisições** Indica a quantidade de requisições processadas com sucesso feitas ao Broker e que não ocorreram erros;
- **Utilização da CPU** Uso médio da CPU do dispositivo Broker;
- **Utilização da memória** Uso médio de RAM do dispositivo Broker;
- **Utilização da memória SWAP** Uso médio da SWAP do dispositivo Broker;

As métricas citadas são essenciais para avaliar a resiliência de um Broker MQTT:

A primeira métrica é essencial para determinar a capacidade do Broker de lidar com cargas de trabalho variáveis e sua capacidade de resposta em situações de alto tráfego. Um tempo de resposta rápido indica eficiência e capacidade de lidar com solicitações rapidamente, o que é importante para manter a resiliência do sistema, especialmente em ambientes com requisitos de tempo real.

Já a segunda mede quantas solicitações o Broker MQTT é capaz de atender em um determinado período de tempo. Uma alta taxa de atendimento indica que o Broker é capaz de lidar com um grande volume de solicitações, o que é fundamental para garantir a resiliência em condições de carga pesada ou ataques de negação de serviço.

Por último, as métricas relacionadas aos subsistemas de hardware como: utilização de CPU, memória e SWAP. Estas são muito importantes para entender a capacidade do Broker MQTT de lidar com cargas de trabalho intensas sem comprometer o desempenho. Um alto uso de CPU pode indicar sobrecarga e potencialmente levar a tempos de resposta mais lentos.

O monitoramento da memória RAM e do espaço de troca é importante para garantir que o Broker tenha recursos suficientes para processar solicitações e evitar falhas por falta de memória.

4.4. Listar parâmetros

Nesta subseção, os parâmetros do sistema e parâmetros de carga de trabalho são listados. Uma lista de parâmetros que podem influenciar o sistema é detalhada nesta subseção. Esses parâmetros são tanto de hardware como de software. A Tabela 3 mostra os parâmetros referentes a carga de trabalho, que são características das solicitações do usuário que variam de um ataque para outro.

Tabela 3 – Carga de trabalho usada nos ataques

Ataque	**Variação de usuários**	***Payload***
Flood CONNECT	100, 500, 1.000 e 10.000	20 bytes
Flood *Payload*	100, 500, 1.000 e 10.000	5 e 20 MB
Flood QoS	1.000	20 bytes

Fonte: O Autor (2024).

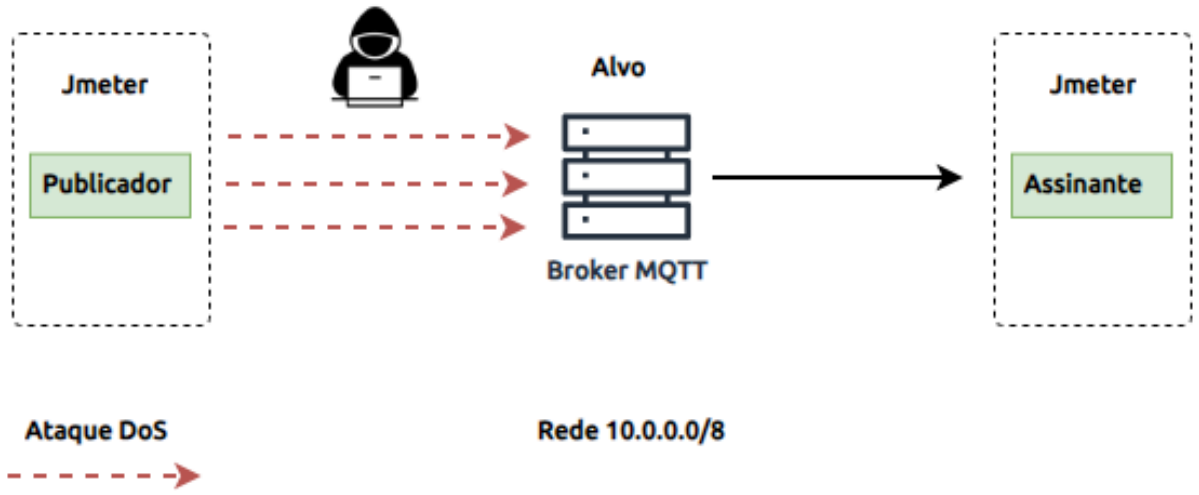
Os testes de carga foram configurados na ferramenta JMeter [69] e instalados em um Core i3 com 4 núcleos, 24 GB de RAM e sistema operacional Windows 11. O Publicador foi emulado no programa JMeter. O assinante usou a seguinte configuração: CPU Core i3-10110U 2,59 GHz (1 núcleo virtual), 1 GB de RAM e sistema operacional, Debian 11 bullseye. Foi usado o cliente mosquitto conhecido como mosquitto sub.

A ferramenta JMeter foi escolhida para realizar os experimentos, por possuir recursos virtuais de usuários *(threads)*, que podem emular um grande número de usuários simultâneos e o tempo de inicialização em segundos que essas mensagens são enviadas para o Broker. O JMeter também tem sido amplamente utilizado em cenários de avaliação de desempenho na área de ciência da computação [69 -71]. Em nossa avaliação, 100, 500, 1.000 e 10.000 usuários simultâneos foram emulados a cada segundo.

O tráfego enviado pelo publicador foi direcionado para a porta 1883, que deve estar aberta para que a informação chegue ao serviço de mensagens e posteriormente seja lida pelo assinante. O recurso padrão do protocolo MQTT é o envio de pacotes com qualidade de serviço em nível 0, comumente chamado “dispare e esqueça”. Ou seja, a informação é enviada, mas não há garantia de entrega. A Tabela 3 apresenta a carga de trabalho dos ataques.

O cenário apresentado na Figura 8 foi utilizado nos testes de ataques de negação de serviço contra os três Brokers MQTT escolhidos. Os Brokers dos experimentos foram instalados em um Raspberry Pi 3 Modelo B com processador Quad Core 1.2GHz com 1GB de RAM. A ferramenta dos experimentos relacionada aos ataques foi instalada em um sistema em ambiente desktop, sistema operacional Windows 11, com processador Intel Core i3 10110U, 24 GB de RAM e uma rede Wi-Fi.

Figura 8 – Cenário dos experimentos



Fonte: O Autor (2024).

O publicador e o assinante foram emulados pelo software JMeter, assim como a quantidade de requisições enviadas aos Brokers MQTT e suas respectivas cargas para realizar os experimentos relacionados ao ataque de negação de serviço.

4.5. Selecionar fatores

A relação de parâmetros pode ser separada em duas categorias: aqueles que passarão por variações durante a avaliação e aqueles que permanecerão constantes. Os parâmetros sujeitos a variações são denominados fatores, e os diferentes valores assumidos por eles são

referidos como níveis. O parâmetro que permaneceu constante na avaliação foi a ferramenta Jmeter.

Já os parâmetros variáveis, assim como seus fatores e níveis escolhidos para este estudo, são:

Fator 1: Carga de Trabalho

- Nível 1: 100 até 10.000 Requisições

Fator 2: Ataques

- Nível 1: Flooding Connect
- Nível 2: Flooding baseado em *payload*
- Nível 3: Flooding baseado em QoS

Fator 3: Brokers

- Nível 1: EMQX
- Nível 2: Mosquitto
- Nível 3: VerneMq

4.6. Selecionar a técnica de avaliação

A escolha da técnica de avaliação está sujeita ao controle eficaz dos parâmetros do sistema, ao tipo de métrica desejada, à disponibilidade de tempo e às ferramentas disponíveis para avaliação e construção do sistema. Este estudo empregará a técnica de "medição", e esta medição será executada no sistema real com o objetivo de extrair os resultados desejados.

4.7. Selecionar carga de trabalho

O sistema sob avaliação precisa ser simplificado para reduzir a complexidade do experimento e ser possível focar nos fatores de interesse, sem comprometer a confiabilidade dos resultados. A carga de trabalho consiste em uma lista de solicitações do serviço ao sistema. Por exemplo, nesta dissertação, na tabela 3 é mostrado a carga de trabalho utilizada para comparar se os Brokers MQTT são resilientes aos ataques negação de serviço e consequentemente, são avaliadas algumas métricas que ajudarão a definir a resiliência destes *Brokers*.

A depender da técnica de avaliação escolhida, a carga de trabalho pode ser expressa de

diferentes formas. Em qualquer uma das técnicas utilizadas, é essencial que a carga de trabalho seja representativa de acordo com a utilização do sistema na vida real. A carga de trabalho, em nossa técnica escolhida, será gerada na prática pelo software JMeter; essa carga será emulada pelo publicador e enviada aos Brokers no momento do experimento.

Definir uma carga de trabalho para avaliar a resiliência dos Brokers MQTT é importante, já que ao impor uma carga significativa, é possível testar a resiliência do Broker MQTT em condições adversas, como picos de tráfego, falhas de rede ou aumento repentino na demanda. Isso ajuda a garantir que o Broker seja capaz de se recuperar eficientemente de situações imprevistas.

Para o ataque *Flooding Connect* a carga foi de apenas 20 bytes, o que para padrões IoT é um tamanho de dado relativamente comum. O propósito dessa carga é junto com a quantidade de usuários simultâneos seja ele 100 ou 10.0000 ver o quão resiliente são os Brokers dos experimentos. Já para o ataque *Flooding* baseado em *payload*, o objetivo é avaliar o quão resiliente são os serviços de mensageria variando a quantidade de dados enviados a eles. Nestes experimentos utilizamos uma carga de 5MB e 20MB. Em relação ao terceiro ataque que é baseado em qualidade de serviço, voltamos ao *payload* para 20 bytes, já que o objetivo da carga neste ataque está relacionado não a quantidade de requisições, mas sim, a variação do QoS.

4.8. Planejar a execução dos experimentos

Esta etapa é de vital importância para garantir resultados mais confiáveis e alinhados com o objetivo do estudo. Ela é determinante, pois é aqui que são definidos todos os parâmetros e procedimentos que serão utilizados para avaliar o desempenho de um sistema, aplicação ou serviço.

4.8.1 Configurações e experimentos

A carga de trabalho a ser empregada foi previamente definida para cada tipo de ataque. Para os ataques de *Flooding Connect* MQTT e *Flooding* baseado em QoS, foi adotada uma carga de 20 bytes. Já para o *Flooding* baseado no tamanho do *payload*, se optou por variar entre as cargas de 5MB e 20MB, conforme ilustrado na Tabela 3 apresentada anteriormente.

4.8.2 Ataques realizados

Os tipos de ataque foram escolhidos com base no documento oficial do OWASP (*Open Web Application Security Project)* [72], que apresenta as 10 maiores vulnerabilidades em sistemas IoT. O OWASP foi escolhido por ser referência mundial na área de segurança computacional. Mais precisamente, os ataques foram escolhidos com base na segunda vulnerabilidade no ranking deste documento OWASP, que é o serviço de rede inseguro.

Os tipos de ataques utilizados estão descritos a seguir:

Flooding Connect É um ataque de inundação de conexão [37] caracterizado pelo envio de muitos pacotes CONNECT para o *Broker* MQTT com solicitações de autenticação. O pacote de conexão é o primeiro pacote que um publicador envia ao Broker para iniciar uma conexão MQTT. Na avaliação, foram avaliadas todas as opções indicadas na Tabela 3 para o ataque de inundação MQTT, utilizando um *payload* de 20 bytes, em cada rodada, por 30 rodadas de experimentos.

Flooding* baseado em *payload Um ataque baseado em carga útil [37] é caracterizado por um invasor ser capaz de esgotar os recursos dos Brokers MQTT enviando mensagens com um grande tamanho de carga útil, resultando em negação de serviço. O protocolo MQTT suporta uma carga útil máxima de 256 MB [47]. Esta avaliação ocorrerá da mesma forma que o ataque anterior, com a mesma ferramenta e a mesma configuração de parâmetros. A única diferença será o aumento da carga útil, que aumentará consideravelmente. A carga de trabalho testada inicialmente será de 5 MB e depois será aumentada para 20 MB. Embora a carga útil seja consideravelmente maior do que o padrão utilizado em mensagens MQTT, elas estão abaixo do tamanho máximo que o protocolo suporta.

***Flooding* baseado em QoS** Um ataque baseado em QoS é caracterizado pelo atacante tentando esgotar os recursos do Broker, enviando um número considerável de mensagens principalmente, com o nível 2 de QoS habilitado. O nível 2 de QoS requer mais recursos computacionais do que os níveis 0 e 1 [37]. Os níveis maiores necessitam de mais trocas de mensagens para confirmar o envio e recebimento das mensagens e assim poder realizar a garantia da comunicação MQTT, como foi possível ver anteriormente na Figura 6. Este ataque será executado da mesma forma que o experimento do ataque de *Flooding Connect*, utilizando a mesma ferramenta e parâmetros de configuração, exceto o fato da modificação nos níveis de QoS. O nível inicial, conhecido como QoS 0, não precisa de modificação, já que

ele é o padrão do protocolo MQTT. Mesmo assim, realizamos os experimentos com o nível 0. Em seguida, o nível 1 e o nível 2 são testados.

Este último nível requer mais poder computacional do que os níveis anteriores, o que é um desafio para sistemas IoT. Esta modificação é realizada no JMeter. O MQTT *Pub Sampler* que representa o publicador e o MQTT *Sub Sampler* que representa o assinante, já que ambos devem receber o mesmo tipo de mensagem de QoS para poder realizar a comunicação.

4.8.3 Detalhamento do ambiente

Os experimentos foram realizados em um ambiente controlado, utilizando os Brokers Mosquitto, EMQX e VerneMQ, instalados em uma plataforma Raspberry Pi. Para realizar os experimentos, foram empregadas ferramentas especializadas, como o JMeter, para emular os ataques de negação de serviço e coletar métricas como taxa de atendimento das requisições e tempo de resposta. Além disso, o Collectl [73] foi utilizado para avaliar o comportamento dos Brokers, e são coletadas informações sobre CPU, memória e SWAP durante os experimentos.

Os ataques serão realizados utilizando a ferramenta JMeter, que pode emular as funcionalidades de microcontroladores e sensores comumente usados em sistemas IoT, emulando assim um ataque de negação de serviço nos Brokers. Inicialmente, foi utilizado um plano de testes criado na ferramenta, e então foi configurado um *thread* de grupo, onde o número de *threads* (usuários virtuais) que deseja se conectar ao Broker de forma simultânea é informado.

O tempo de inicialização, em segundos, também é configurado no JMeter quando os usuários enviam solicitações de conexão MQTT para os Brokers. Então, o amostrador de conexão MQTT é utilizado para a criação de uma conexão. Neste ponto, é informado o endereço IPV4 ou IPV6 do Broker, a porta a qual ele está escutando e a versão do protocolo que será configurada. Por fim, também foi configurado o amostrador MQTT *Pub Sampler*, que emula, por exemplo, publicadores, tópicos, determina a carga útil da mensagem e os níveis de QoS a serem enviados.

Na emulação, o publicador e o assinante se conectam ao *Broker* através do JMeter [39], que simula a comunicação e as cargas de trabalho utilizadas no ataque, assim como a variação nos níveis de QoS. Esta ferramenta é amplamente adotada pela comunidade,

principalmente, no contexto de testes de carga de trabalho.

- O primeiro ataque realizado foi o ataque de inundação contra o protocolo MQTT e com uma carga de 20bytes;
- O segundo ataque foi o mesmo, mas a carga útil da mensagem foi aumentada para 5 MB e 20 MB;
- O terceiro ataque também se baseou em inundações, mas com diferentes tipos de QoS (0,1,2) e com carga útil de 20 bytes.

4.8.4 Realização dos experimentos

Para aprimorar a confiabilidade dos resultados, foram realizadas trinta repetições para cada configuração de carga de trabalho, variação de usuários e tipos de ataque. No entanto, uma exceção foi feita para o ataque baseado em QoS, onde o foco principal reside na avaliação da resiliência do Broker em relação aos diferentes níveis de QoS. Nesse caso, a estratégia adotada foi isolar outros fatores para cada nível de qualidade de serviço e para cada Broker envolvido.

Para a realização dos experimentos, foram necessárias trezentas e sessenta tentativas em cada um dos ataques (exceto o ataque de *Flooding* QoS, que teve uma quantidade menor de tentativas, pois o objetivo é avaliar como o Broker se comporta diante do ataque com níveis distintos de QoS). Foram realizados cento e vinte experimentos relacionados ao ataque *Flooding connect* em cada um dos *Brokers*, onde foram feitos trinta experimentos para cada variação de usuários. A Tabela 4 detalha estas informações.

Tabela 4 – Quantidade de experimentos no ataque *Flooding connect, Flooding* baseado no *payload* de 5MB e 20MB

Tipo de ataque / Broker	**Variação de usuários**	**Experimentos**	**Total**
Ataque Flooding / EMQX	100, 500, 1.000, 10.000	30	30 * 4 variação de usuários = 120 experimentos
Ataque Flooding / Mosquitto	100, 500, 1.000, 10.000	30	30 * 4 variação de usuários = 120 experimentos
Ataque Flooding / VerneMQ	100, 500, 1.000, 10.000	30	30 * 4 variação de usuários = 120 experimentos

Fonte: O Autor (2024).

O mesmo número de experimentos foi realizado para o ataque de *Flooding* baseado no *payload* de 5 MB. Assim como, a mesma quantidade de trezentos e sessenta experimentos para o ataque de *Flooding* baseado no *payload* de 20 MB, a Tabela 4 pode ser utilizada para entender a quantidade de experimentos realizados. E, por fim, foram realizados duzentas e setenta tentativas para o ataque de *Flooding* QoS, utilizando os três níveis (0, 1 e 2), onde cada *Broker* realizou trinta tentativas de cada experimento relacionado ao nível de QoS utilizado. A Tabela 5 apresenta, de forma resumida, este cálculo.

Tabela 5 – Quantidade de experimentos no ataque *Flooding* baseado em níveis de QoS

Tipo de ataque / Broker	**Variação de usuários**	**Experimentos**	**Total**
Ataque Flooding QoS (0,1 e 2) / EMQX	1000	30	30 * 3 (níveis de QoS) * 1 tipo de requisição = 90 experimentos
Ataque Flooding QoS (0,1 e 2) / Mosquitto	1000	30	30 * 3 (níveis de QoS) * 1 tipo de requisição = 90 experimentos
Ataque Flooding QoS (0,1 e 2) / VerneMQ	1000	30	30 * 3 (níveis de QoS) * 1 tipo de requisição = 90 experimentos

Fonte: O Autor (2024).

No início, observou-se que, na maioria dos experimentos, os ataques de negação de serviço duraram no máximo sessenta segundos. Já em relação a quantidade de usuários nos experimentos, ao atingirem 10.000 solicitações por segundo, o hardware do Broker ficou indisponível por muitas vezes. Para garantir uma maior confiabilidade nos resultados experimentais, foi estabelecido que aconteceria 30 amostras independentemente dos resultados, em cada um dos ataques e com suas respectivas variações de carga e usuários.

Para cada experimento realizado em cada um dos Brokers, foi assegurado que o serviço de mensageria e o hardware não fossem afetados pelo consumo de recursos do experimento anterior. Sendo assim, após cada amostra dos experimentos, o sistema foi reinicializado e aguardou-se cento e vinte segundos, garantindo que o sistema operacional, o hardware e o serviço de mensagens estivessem estáveis antes de prosseguir com as próximas rodadas de testes.

Por fim, o número total de experimentos chegou a 1.350 testes, e este esforço tem como objetivo avaliar a resiliência dos Brokers MQTT frente a ataques de negação de serviço.

4.9. Analisar e interpretar os dados

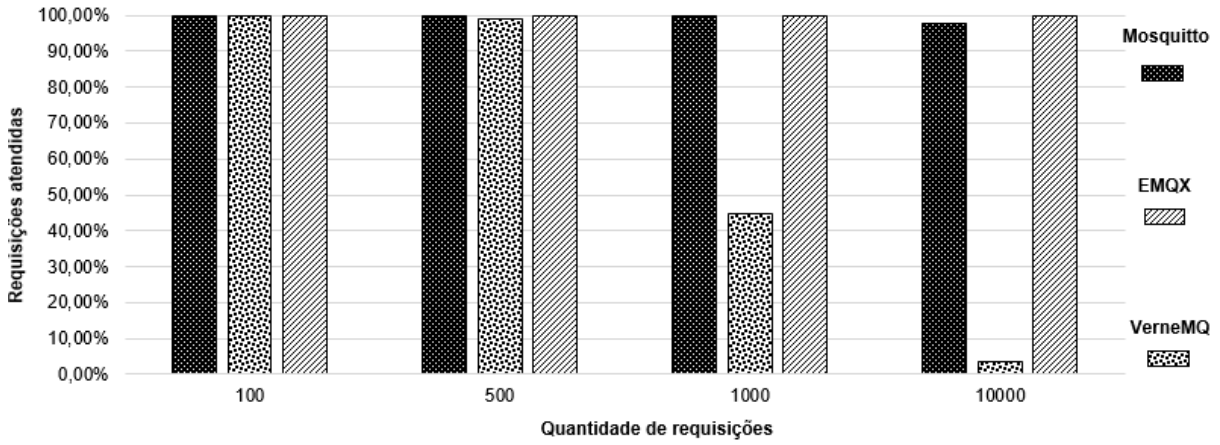
A análise e interpretação dos resultados obtidos é importante para a obtenção de *insights* valiosos e orientar decisões estratégicas. Através desta análise, é possível identificar áreas de destaque, oportunidades de melhoria e tendências ao longo do tempo, contextualizando os resultados para uma compreensão mais profunda dos fatores que afetam o desempenho. Já a interpretação é essencial para compreender que a análise produz somente resultados, mas não conclusões imediatas. Esses resultados estabelecem a base a partir da qual os tomadores de decisão podem formar conclusões devidamente informadas.

Esta seção analisa e interpreta os resultados dos experimentos no cenário proposto. Inicialmente, foram avaliados os resultados dos Brokers quanto ao seu comportamento diante de ataques de negação de serviço: *Flooding CONNECT*, *Flooding* baseado em carga útil e *Flooding* baseado em QoS.

4.9.1 Resultados do Ataque *Flooding Connect*

Esta subseção resume os resultados do ataque de *Flooding Connect*. A Figura 9 apresenta os resultados em relação a taxa de atendimento de requisições.

Figura 9 – Taxa de atendimento de requisições com ataque *Flooding Connect*



Fonte: O Autor (2024).

Através dos resultados relacionados a taxa de atendimento de requisições, é possível saber o resultado da taxa de erro, que são as requisições não atendidas pelo Broker. Nos testes com 100 requisições, todos os Brokers se comportaram adequadamente, sem nenhuma porcentagem de erros. Já para os testes com 500 requisições, tanto o EMQX como o

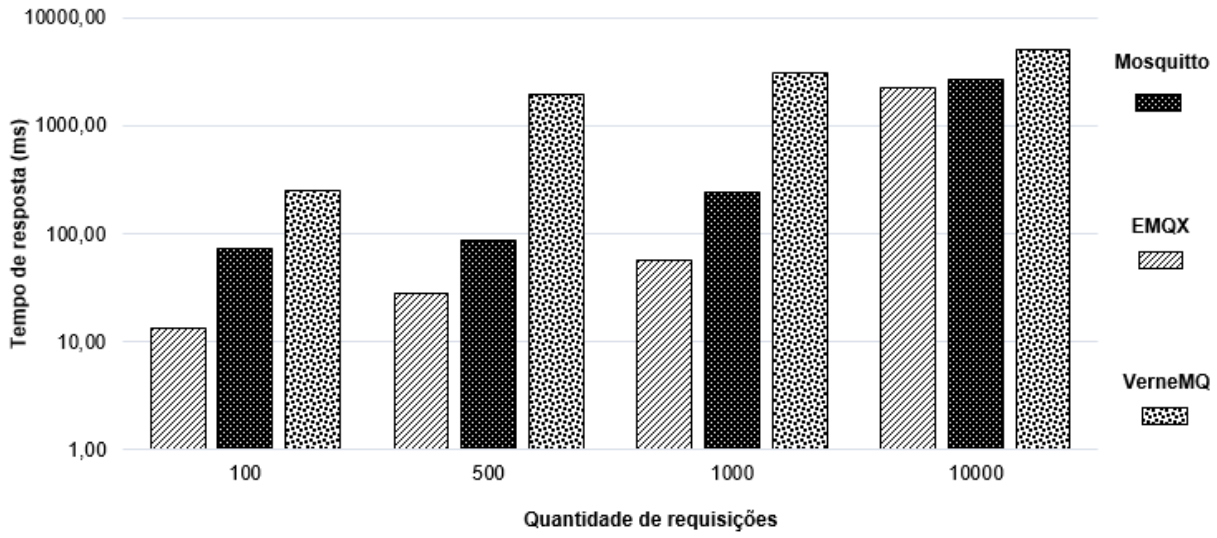
Mosquitto responderam com sucesso a todas requisições; contudo, o VerneMQ apresentou alguma instabilidade, chegando a uma taxa de erro de quase 1%, consequentemente, atendendo a 99% das requisições enviadas ao Broker.

Para os experimentos com 1.000 requisições, os resultados tanto do EMQX quanto do Mosquitto não se alteraram em relação a taxa de atendimento das requisições. Ambos conseguiram atender todas as requisições. Por outro lado, em relação ao VerneMQ, a quantidade de requisições não atendidas aumentou significativamente, chegando a uma taxa de erro de mais de 55%.

Por fim, em relação ao maior número de requisições, houve uma mudança significativa nos resultados. Os Brokers EMQX e Mosquitto até então estavam atendendo 100% das requisições. O EMQX teve uma taxa de erro de 0,27%, enquanto o Mosquitto teve uma taxa de erro de 2,25% no atendimento das requisições.O VerneMQ continuou apresentando mais instabilidade que os outros, atendendo apenas 3,61% das suas requisições.

Os resultados não chegam a serem distantes entre si, os quais podem ser explicados pelo desvio padrão que estão nos Apêndices A, B e C desta dissertação. Como os resultados dos desvios padrões estão próximos da média, é possível afirmar que existe uma menor variabilidade dos dados, sugerindo que os dados obtidos estão dentro de uma margem de erro aceitável. Os resultados relacionados ao tempo de resposta estão disponíveis na Figura 10.

Figura 10 – Tempo de resposta com ataque *Flooding Connect* (ms)



Fonte: O Autor (2024).

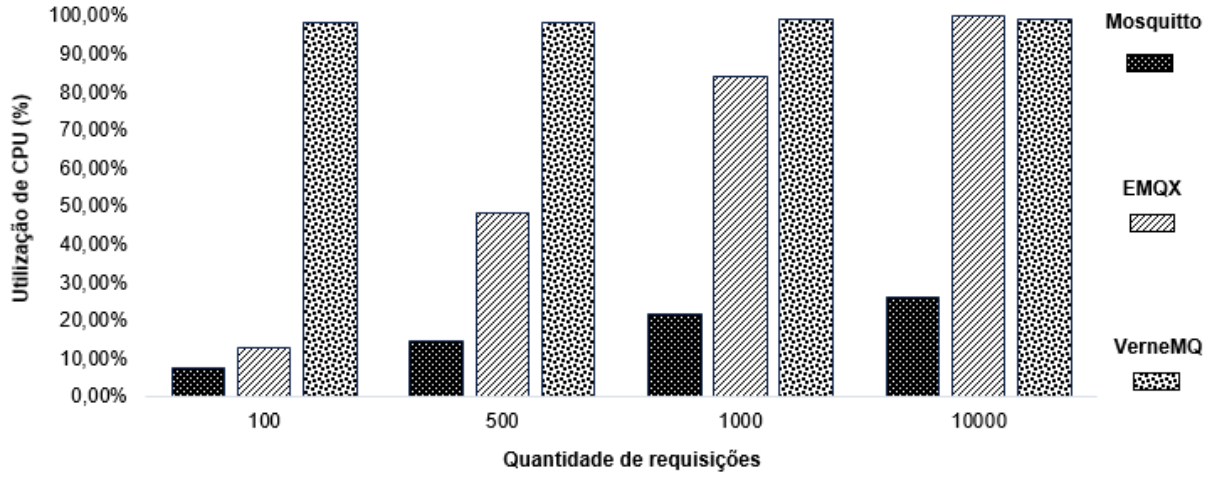
É possível perceber que, com o aumento das solicitações, o tempo de resposta aumenta proporcionalmente em todos os Brokers. O Broker que teve o melhor tempo de resposta foi o EMQX; ele teve o melhor resultado em relação aos outros em todas as quantidades de requisições. Para 100 solicitações, ele teve um tempo de resposta médio de 13,10 ms, enquanto que para 10.000 solicitações teve média de 2.231,97 ms.

Enquanto isso, o VerneMQ teve o pior desempenho, com 249,33 ms para o menor número de solicitações e 5.081,90 ms para o maior número de solicitações. Já o Mosquitto ficou entre os dois, com tempo de resposta de 73,27 ms para 100 solicitações e 2.656,70 para o maior número de solicitações.

Observou-se também que, para 100 solicitações, o VerneMQ teve um aumento no tempo de resposta de mais de 1.800% em comparação com EMQX e 240% quando comparado ao Mosquitto.

Já em relação às métricas relacionadas aos subsistemas dos Brokers, como processador, memória RAM e memória SWAP, o Broker que teve o melhor desempenho para o ataque Flooding Connect foi o Mosquitto. Este Broker não chegou a consumir 26% de CPU no número máximo de requisições, conforme ilustrado na Figura 11.

Figura 11 – Utilização de CPU no ataque *Flooding Connect*

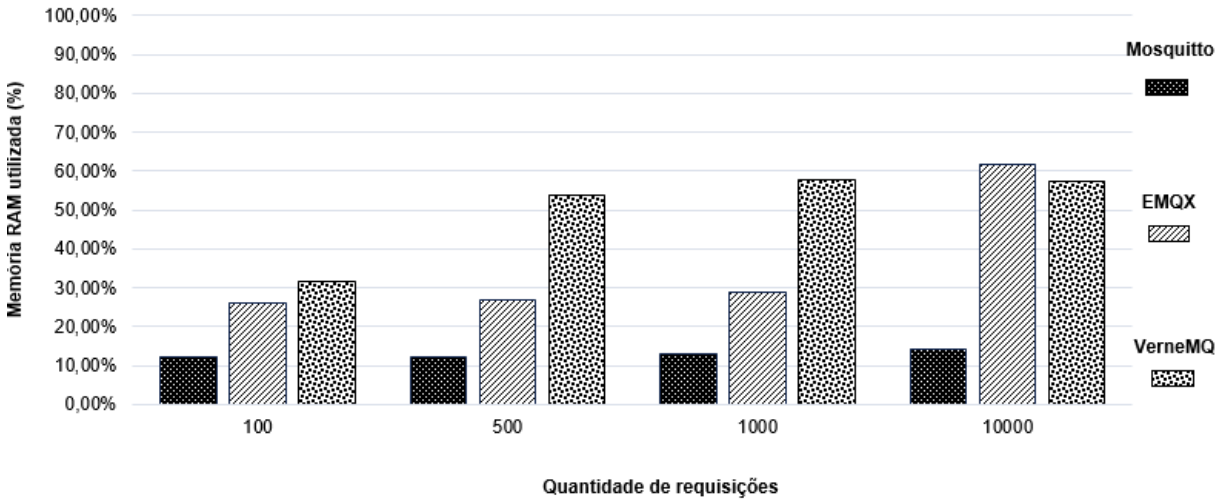


Fonte: O Autor (2024).

O mesmo padrão ocorreu na utilização da memória. O Broker Mosquitto não chegou a consumir nem 15% de sua memória principal, conforme mostrado na Figura 12. Este

consumo continuou estável mesmo quando o Broker foi testado para o maior número de solicitações. Este Broker também não utilizou memória de disco em nenhum momento dos ataques *Flooding Connect.*

Figura 12 – Utilização de memória RAM no ataque *Flooding Connect*



Fonte: O Autor (2024).

Por sua vez, o VerneMQ foi significativamente afetado, consumindo cerca de 98,90% de utilização do seu processador e 58% de sua memória RAM. Devido a esta situação, uma grande parte da memória do disco, conhecida como SWAP, foi utilizada, atingindo 32% de uso para a quantidade máxima de requisições, degradando significativamente o desempenho do *Broker* que, por muitas vezes, fez o serviço de mensageria ficar indisponível e o *Broker* teve que ser reiniciado.

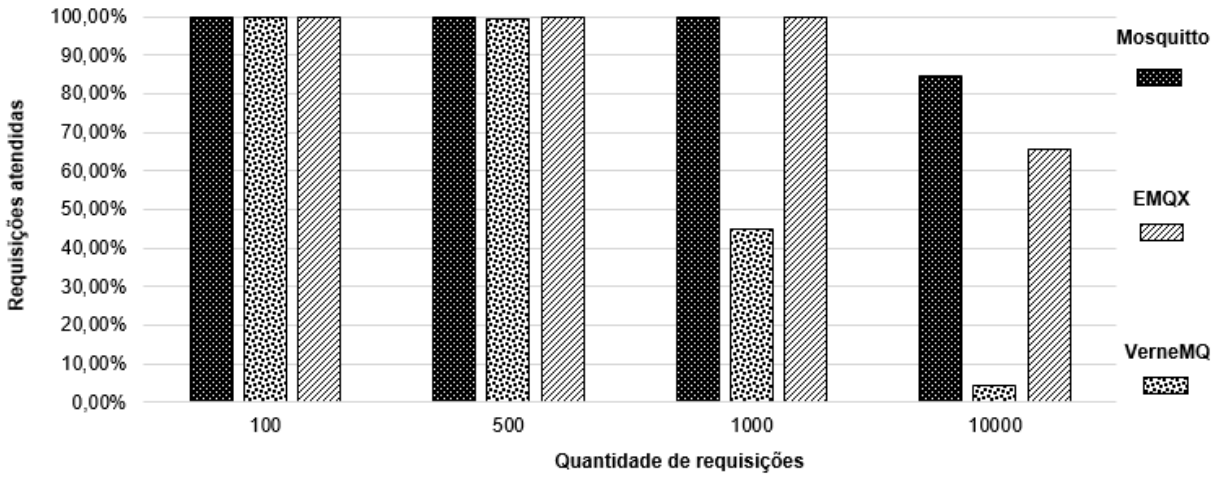
A respeito do EMQX, teve um aumento considerável no consumo relacionado ao processamento a partir de 1.000 conexões, onde atingiu picos de 84% de utilização e aumentou para 99,83% no experimento seguinte. Em relação ao consumo de sua memória principal, chegou a atingir picos de 61,93% para 10.000 solicitações. Assim como o Mosquitto, o EMQX em nenhum momento precisou utilizar a memória SWAP, evitando assim a degradação do serviço e consequentemente a indisponibilidade do serviço de mensageria.

4.9.2 Resultados do Ataque *Flooding* baseado em *payload*

Em relação aos resultados do segundo ataque, *Flooding* baseado em *payload*, com a utilização de 5MB de carga útil, em termos de atendimento das requisições, o resultado ficou

dentro do esperado. Quanto mais se aumentava o número de requisições, mais a possibilidade de atendimento das requisições diminuía proporcionalmente em todos os Brokers, consequentemente, gerando uma taxa de erro maior. O que está ilustrado na Figura 13.

Figura 13 – Taxa de atendimento de requisições com ataque *Flooding* baseado em *payload* de 5MB



Fonte: O Autor (2024).

O ataque de *payload* de 5MB e o de *Flooding Connect* não tiveram problemas em relação a quantidade mínima de requisições para todos os Brokers. Eles atenderam todas as requisições sem nenhuma porcentagem de erros. Um resultado esperado, já que todos os Brokers testados atendem pelo menos 100 requisições por segundo.

Já em relação aos testes com 500 requisições, tanto o EMQX como o Mosquitto continuaram atendendo a todas as requisições. O VerneMQ teve um resultado bem parecido com o ataque anterior, chegando a uma taxa de erro de quase 1%; ou seja, das quinhentas requisições, cinco delas não chegaram ao Broker.

Para os experimentos com 1.000 requisições, os resultados tanto do EMQX quanto do Mosquitto não se alteraram em relação a taxa de erro, que continuou em 0%. Ou seja, ambos atenderam a todas as solicitações realizadas. Quanto ao VerneMQ, a quantidade de requisições não atendidas continuaram aumentando significativamente, chegando a uma taxa de erro de mais de 550 requisições não atendidas pelo Broker em questão.

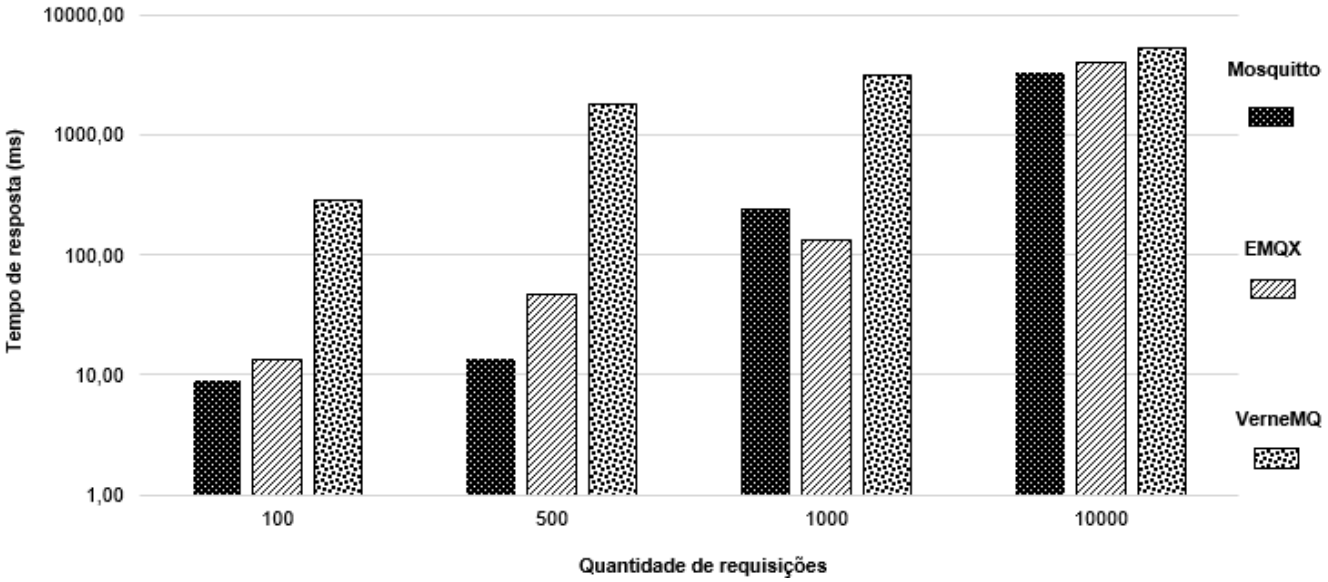
Finalmente, para os testes de 10.000 requisições, houve uma mudança importante nos resultados, principalmente nos Brokers EMQX e no Mosquitto, pois ambos até então estavam atendendo a todas as requisições realizadas para este tipo de ataque. O Broker EMQX atendeu

apenas 65,76% das suas requisições, enquanto que o Mosquitto se saiu melhor e conseguiu 84,56% no atendimento de suas requisições. Ou seja, das 10.000 requisições o Mosquitto atendeu 8.456 requisições, um número 28,6% maior que o EMQX.

Já o VerneMQ continuou atendendo um número reduzido de requisições; a taxa de erro para a maior quantidade de requisições chegou a próximo de 96%. Este Broker atendeu uma média de 400 requisições apenas.

Os resultados do ataque baseado em *payload* com carga útil de 5 MB, em termos de tempo de resposta, foi o esperado. Quanto mais se aumentava o número de requisições, o tempo de resposta aumentava proporcionalmente em todos os *Brokers*, conforme ilustrado na Figura 14.

Figura 14 – Tempo de resposta do ataque *Flooding* com *payload* de 5MB (ms)



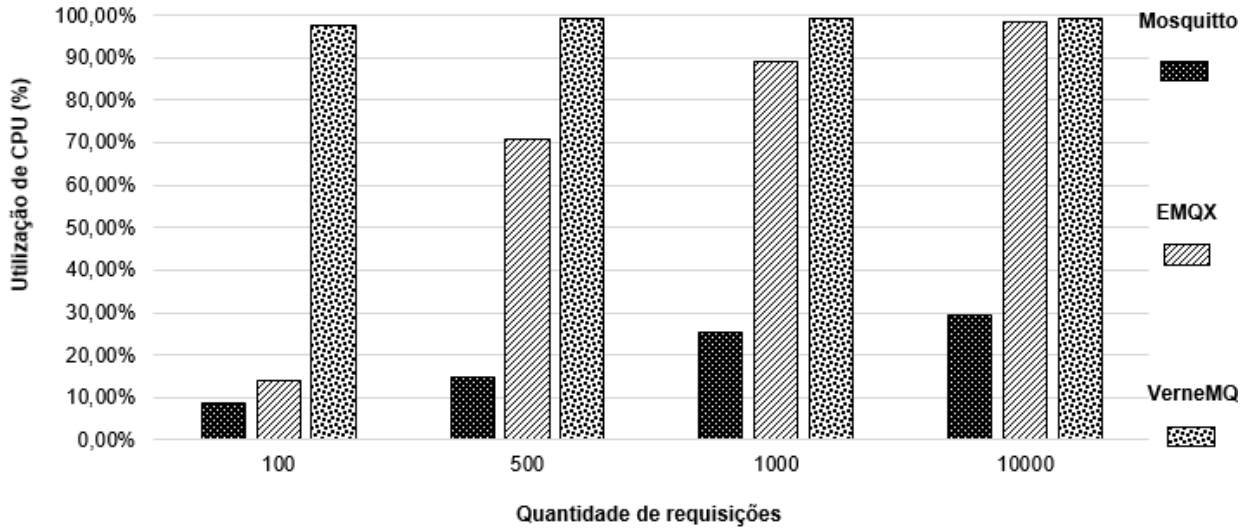
Fonte: O Autor (2024)

O tempo de resposta no *Broker* EMQX para a quantidade mínima de requisições foi, em média, 13,37 ms, chegando a 4.015,67 ms quando o número de solicitações foi 10.000. Situação similar foi observada com o VerneMQ. No primeiro teste, obteve-se um tempo de resposta de 288,60 ms, um resultado bem acima do tempo de resposta do EMQX para o mesmo número de solicitações. Em relação ao maior número de solicitações, o VerneMQ obteve um tempo de resposta médio de 5.251,20 ms, um aumento de quase 31% em relação ao EMQX considerando10.000 requisições.

O Broker VerneMQ, por muitas vezes, não estava disponível para uso, quando testado para o maior número de requisições, gerando assim negação de serviço. Neste ataque, o Broker Mosquitto teve um melhor desempenho em termos de tempo de resposta em comparação com os outros Brokers. Para 100 solicitações, obteve um tempo médio de 8,93 ms; já para o máximo de 10.000, obteve um tempo médio de 3.283,50 ms. Ou seja, um tempo de resposta de quase 18% melhor que o EMQX considerando a quantidade máxima de requisições.

Em relação às métricas relacionadas ao hardware, o Broker com melhor desempenho continuou sendo o Mosquitto. Este Broker não consumiu nem 30% de CPU para o maior número de requisições, como mostra a Figura 15.

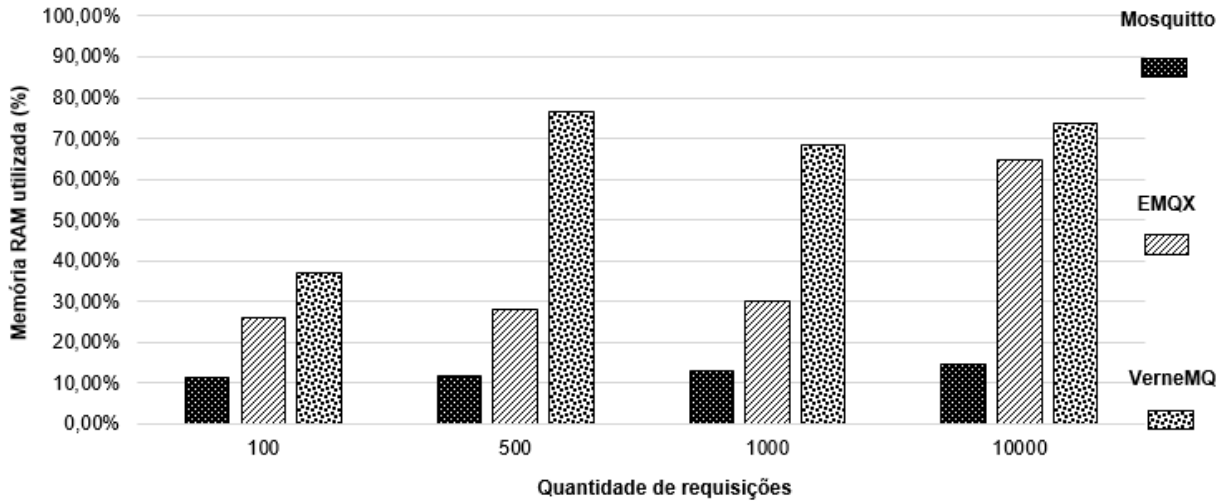
Figura 15 – Utilização de CPU do ataque *Flooding* com *payload* de 5 MB



Fonte: O Autor (2024).

O mesmo aconteceu com a memória principal, que não chegou a consumir nem 15% do seu total, conforme mostrado na Figura 16. Também não houve utilização da memória de disco para nenhuma quantidade de requisições testadas. Por outro lado, o VerneMQ foi consideravelmente afetado, consumindo em média 98,90% do seu processamento em todas as solicitações e 65% de sua memória RAM.

Figura 16 – Utilização de memória RAM do ataque *Flooding* com *payload* de 5 MB



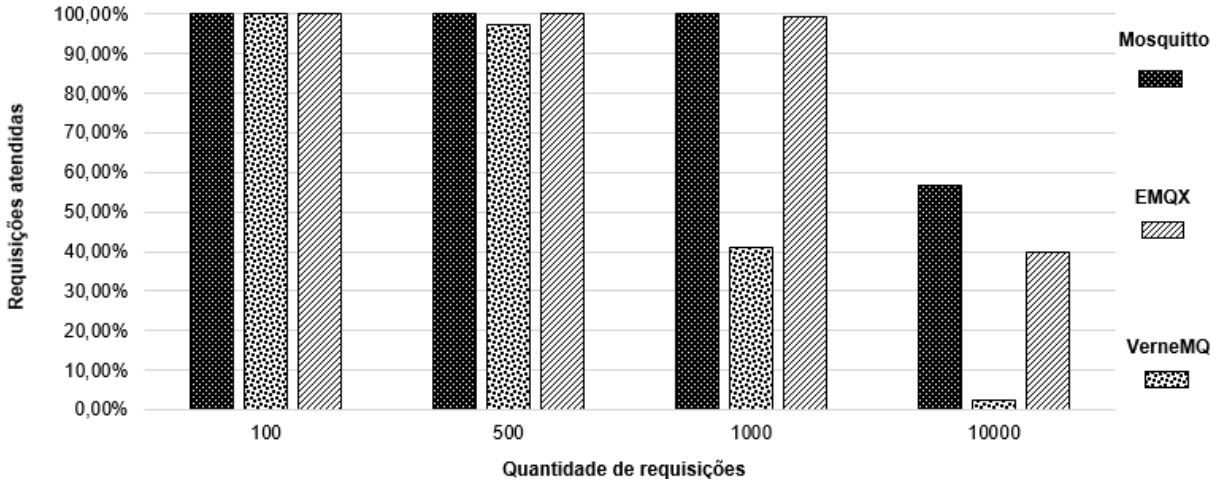
Fonte: O Autor (2024).

Além disso, mais uma vez, uma parte da memória de disco foi muito utilizada, começando em 25,87% de utilização para 500 requisições e chegando a quase 67% de utilização para a maior quantidade de requisições, degradando o serviço de mensageria. Por muitas vezes, foi necessário reiniciar o equipamento, gerando inclusive a negação de serviço.

Os resultados do EMQX foram de 14% na utilização do processamento para uma quantidade de 100 solicitações e aumentou para 71% para 1.000 solicitações, atingindo picos de 98,53% para o maior número de solicitações. Já em relação a memória RAM, o EMQX utilizou um máximo de 64,81% de utilização para 10.000 requisições e pela primeira vez, este Broker chegou a utilizar memória de disco, algo em torno de 2,26% de utilização. Contudo, esta baixa utilização não chegou a deixar o sistema indisponível.

Os resultados dos Brokers em relação ao ataque de carga útil de 20 MB, conforme ilustrado na Figura 17, seguiu o mesmo padrão em relação a taxa de atendimento das requisições. Ou seja, continuou aumentando a taxa de erro de acordo com o aumento da carga de trabalho maior.

Figura 17 – Taxa de atendimento de requisições com ataque *Flooding* baseado no *payload* de 20MB



Fonte: O Autor (2024).

No ataque de *payload* de 20MB, novamente não ocorreram problemas em relação a quantidade mínima de requisições, todos os Brokers atenderam a todas solicitações sem apresentar erros em nenhum momento. A mesma situação ocorreu com os experimentos de 500 requisições; tanto o EMQX como o Mosquitto continuaram atendendo a todas as requisições. Contudo, o VerneMQ continuou aumentando a sua taxa de erro, chegando a quase 3%; ou seja, das quinhentas requisições, quinze delas não chegaram ao Broker.

Para a avaliação de 1.000 requisições no ataque de *payload* de 20MB, houve uma diferença significativa em relação aos resultados do EMQX, pois pela primeira vez, para essa quantidade de requisições, o mesmo não conseguiu atender a todas as demandas. Ele teve uma taxa de erro de aproximadamente 0,6%. Já os resultados do Mosquitto não se alteraram em relação a taxa de erro, ele continuou com taxa de 0%

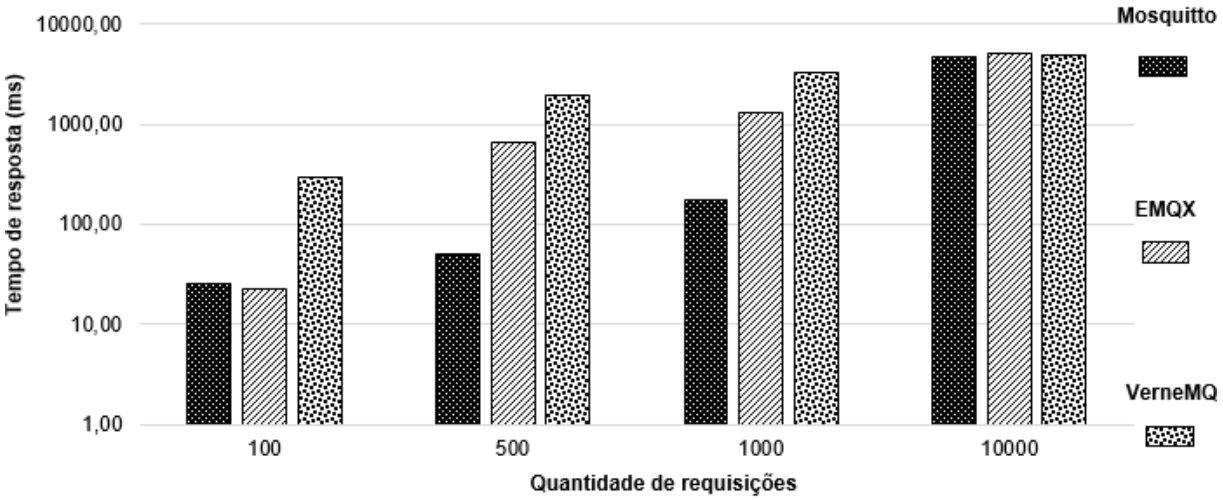
Quanto ao VerneMQ, a quantidade de requisições não atendidas continuaram aumentando significativamente, chegando a uma taxa de erro de quase de 59%. Quando os experimentos foram para a maior quantidade de requisições, todos os Brokers deixaram de atender às suas requisições por completo. O EMQX teve uma taxa de erro de 60,14%, a maior até então para este Broker, fazendo com que o mesmo só conseguisse atender 3.986 solicitações. Por sua vez, o Mosquitto se saiu um pouco melhor e teve uma taxa de erro menor em comparação ao EMQX, chegando a 43,13%, atendendo assim 5.687 requisições, das 10.000 enviadas ao Broker. Já o VerneMQ continuou com uma taxa de erro alta, chegando a quase 98%; ou seja, das 10.000 requisições realizadas, este Broker só conseguiu atender

aproximadamente 200 requisições.

Em relação ao tempo de resposta, os resultados se encontram ilustrados na Figura 18. O tempo de resposta continuou aumentando em todas as opções de requisições para este tipo de ataque com uma carga de trabalho maior, como já era esperado. No entanto, houve um aumento significativo nos tempos de resposta. O EMQX, por exemplo, para 100 solicitações, teve um tempo médio de 22,90 ms, atingindo 5.160,57 ms para 10.000 solicitações.

Por sua vez, o VerneMQ, para o primeiro teste, teve um tempo de resposta de 297,97 ms, e para o maior número de solicitações teve um resultado muito próximo do EMQX. Finalmente, o Mosquitto teve um tempo médio de resposta menor para todas as solicitações em comparação com os outros Brokers.

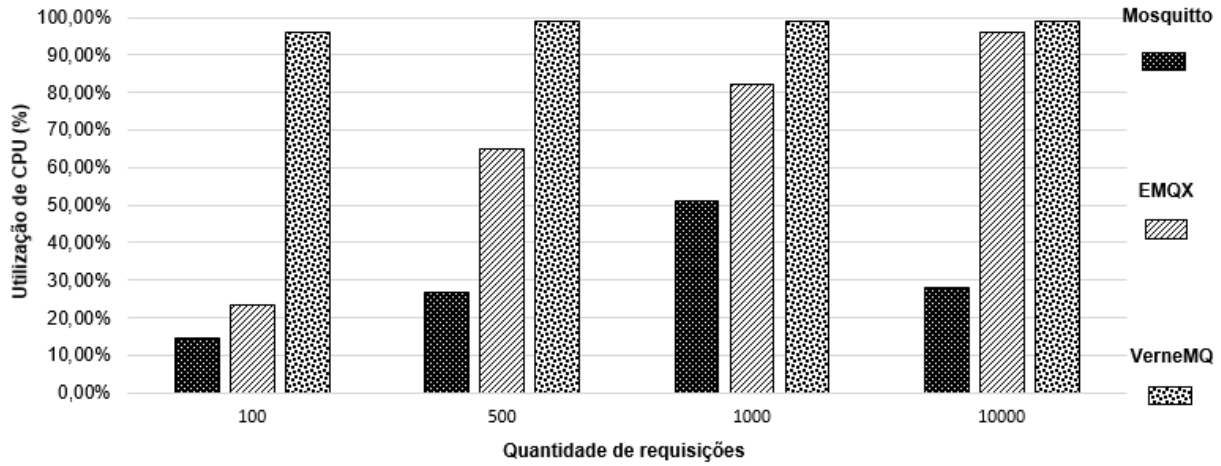
Figura 18 – Tempo de resposta do ataque *Flooding* com *payload* de 20MB (ms)



Fonte: O Autor (2024).

Em relação às métricas relacionadas ao hardware, o Mosquitto continuou tendo os melhores resultados, apesar dos picos de 51% na utilização do seu processamento, o mais alto até então, conforme mostrado na Figura 19.

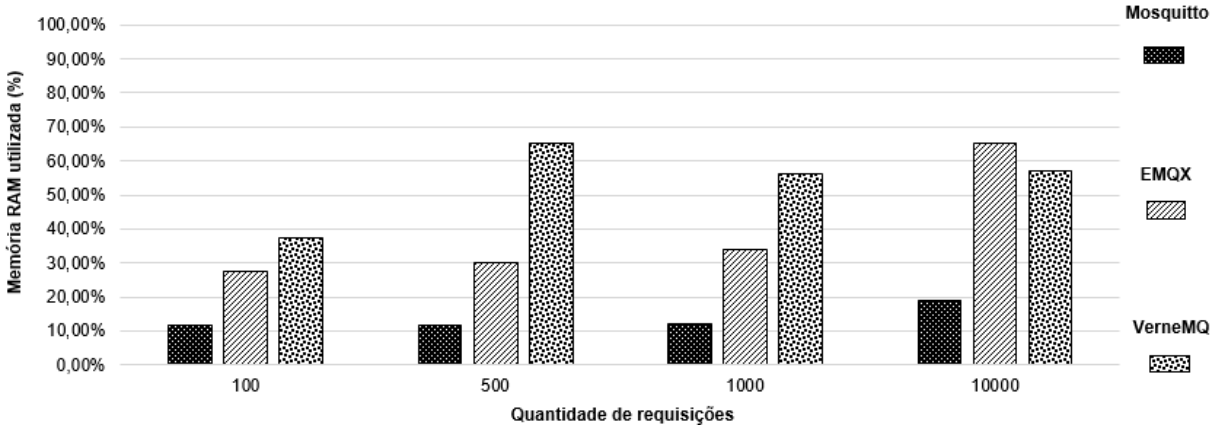
Figura 19 – Utilização de CPU do ataque *Flooding* com *payload* de 20 MB



Fonte: O Autor (2024).

O uso de sua memória principal também aumentou, conforme ilustrado na Figura 20, mas não ultrapassou 19% de uso. Já em relação à memória SWAP, o Mosquitto não chegou a utilizar a mesma para nenhuma quantidade de requisições. Este é um bom sinal, já que a utilização de memória de disco degrada muito o desempenho dos sistemas e, por muitas vezes, acaba gerando a necessidade de reiniciar o equipamento.

Figura 20 – Utilização de memória RAM do ataque *Flooding* com *payload* de 20 MB



Fonte: O Autor (2024).

O VerneMQ continuou a ser significativamente afetado, atingindo picos de processamento de cerca de 99,13% em média para todas as requisições e o consumo da

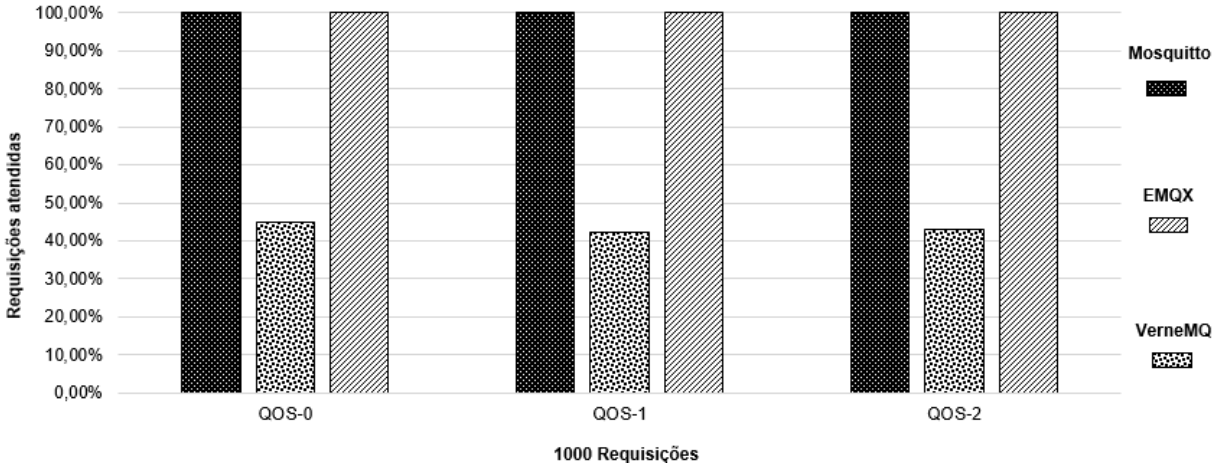
memória principal chegando a uma utilização média de 65% de sua capacidade. O disco, mais uma vez, foi bastante utilizado, consumindo muita memória SWAP e degradando o serviço de mensagens.

Novamente, o equipamento teve que ser reiniciado, pois o mesmo não conseguiu se recuperar da alta carga de dados e requisições, gerando assim negação de serviço. Esperou-se em média cento e vinte segundos para que o sistema como um todo fosse restabelecido. Já o EMQX teve um desempenho de 25,65% de uso de memória RAM para 100 solicitações e, para o máximo de solicitações, uma utilização média de 65,10%. Em relação a memória de disco continuou tendo pouca utilização para a quantidade máxima de requisições.

4.9.3 Resultados do Ataque *Flooding* baseado em QoS

Finalmente, temos os resultados relacionados ao ataque Flooding baseado em qualidade de serviço (e seus níveis). Foram avaliados todos os níveis: o QoS 0, que é o padrão do protocolo MQTT, o QoS 1 e o QoS 2. Neste ataque, só foram testados um valor padrão de requisições para os diferentes níveis de QoS. Os resultados tanto do EMQX quanto do Mosquitto não se alteraram em relação a taxa de atendimento das requisições, como mostra a Figura 21. Ambos conseguiram atender todas as requisições. Quanto ao Broker VerneMQ, a quantidade de requisições não atendidas continuaram altas, em torno de 57%.

Figura 21 – Taxa de atendimento de requisições do ataque *Flooding* com níveis distintos de QoS



Fonte: O Autor (2024).

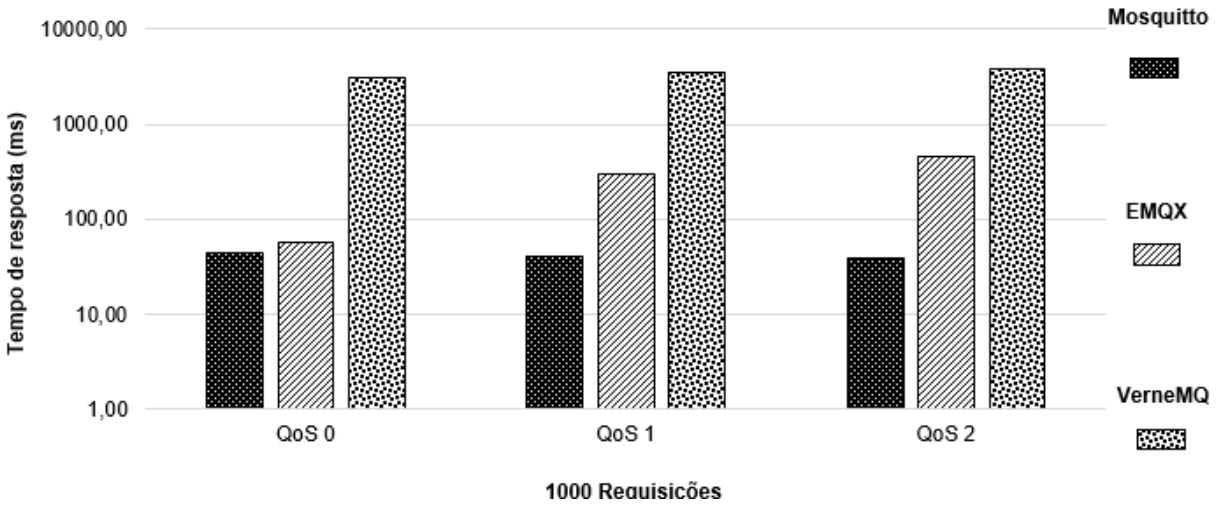
Em relação ao tempo de resposta, o EMQX para o QoS 0 teve um tempo médio de 56,60. Em relação ao QoS 1, foi observado um aumento de quase 450% em relação ao tempo

de resposta do QoS 0. Finalmente, em relação ao QoS 2, esse tempo foi ainda maior, atingindo 451,70 ms, um aumento de quase 50% em relação ao QoS 1.

No VerneMQ, o tempo de resposta aumentou consideravelmente em comparação com EMQX. Por exemplo, para o primeiro nível de QoS, foi observado um tempo de resposta de 3.133,57 ms. Em relação ao QoS 1, o tempo de resposta foi 15% superior ao primeiro, enquanto que a avaliação com QoS 2 apresentou um aumento de quase 5% em relação ao QoS 1, atingindo 3.772,53 ms.

Mais uma vez, o Mosquitto teve um tempo de resposta relativamente baixo para todos os níveis de QoS, conforme ilustrado na Figura 22. O tempo de resposta foi em média de 41,40 ms para obter uma resposta deste Broker, o que levou o mesmo a ter o melhor desempenho entre os Brokers avaliados neste quesito

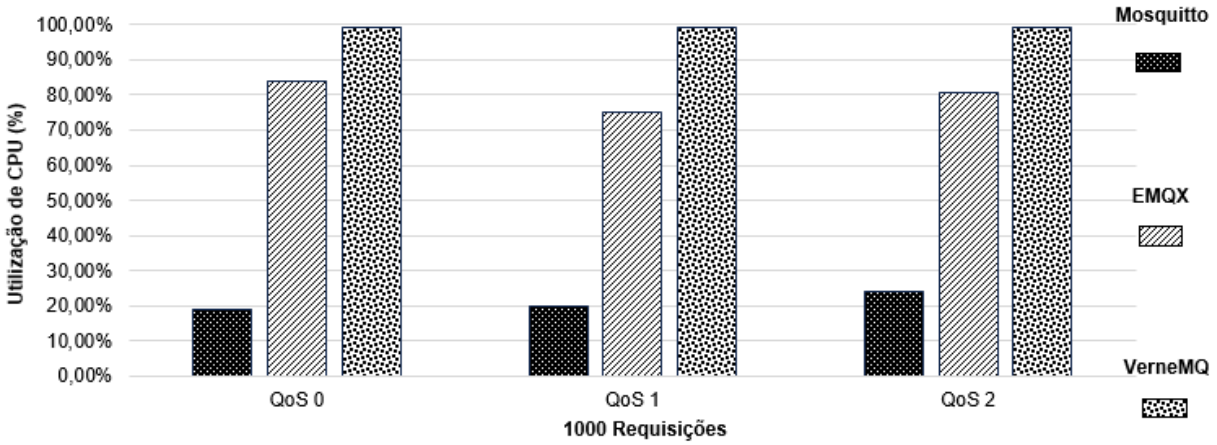
Figura 22 – Tempo de resposta do ataque *Flooding* com níveis distintos de QoS (ms)



Fonte: O Autor (2024).

Em relação ao hardware, o Mosquitto é o Broker que melhor gerenciou o processamento das solicitações, conforme ilustrado na Figura 23. Assim como, a parte de gerenciamento de memória, conforme mostrado na Figura 24.

Figura 23 – Utilização de CPU do ataque *Flooding* com níveis distintos de QoS

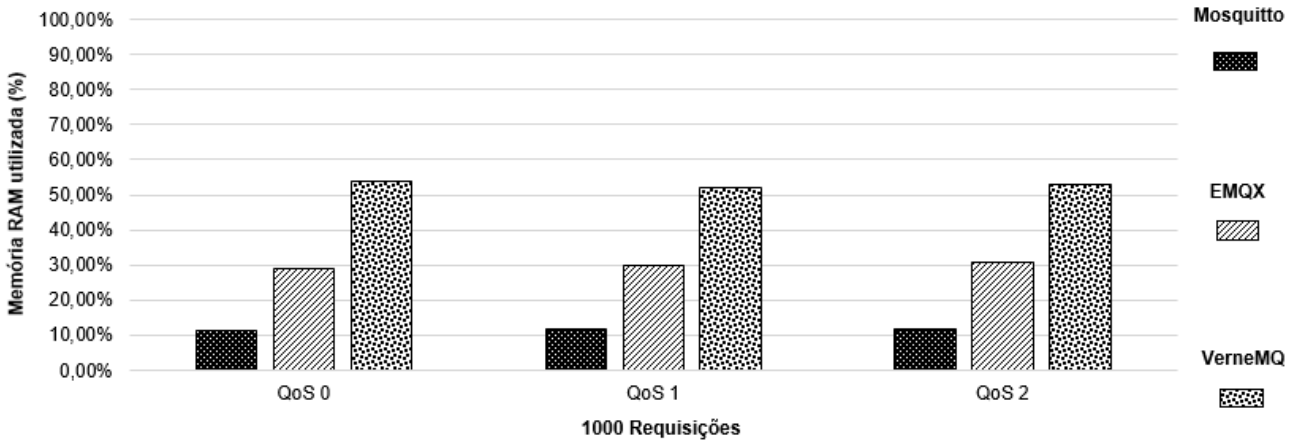


Fonte: O Autor (2024).

O Mosquitto continuou tendo o melhor desempenho entre os Brokers. O VerneMQ continuou a ser "punido" em relação ao hardware, pois os picos de processamento mantiveram uma utilização média de 99% para todas as requisições e a memória principal foi fortemente afetada. Em relação a SWAP, continuou-se a usar espaço em disco, degradando ainda mais o desempenho do Broker, muitas vezes deixando-o indisponível.

Finalmente, o EMQX que teve um alto consumo de CPU, atingindo uma média de 79,86% de uso para todas as requisições. Contudo, a memória RAM não foi muito afetada, pois ela não ultrapassou 30,5% de utilização. Consequentemente, não foi necessário usar a memória SWAP, evitando a perda de mensagens enviadas ao Broker.

Figura 24 – Utilização de memória RAM do ataque *Flooding* com níveis distintos de QoS



Fonte: O Autor (2024).

4.10 Apresentar os resultados

Na fase conclusiva de estudos de desempenho, segundo Jain [17], é fundamental comunicar os resultados. Este passo final envolve compartilhar os resultados de maneira clara e compreensível, permitindo que os gestores compreendam os mesmos e, com base nisso, tomem decisões nos seus projetos.

De forma a melhor apresentar os resultados evidenciados na seção anterior, esta seção busca evidenciar os mesmos de forma sumarizada. Para o primeiro ataque, *Flooding Connect*, a taxa de atendimento das requisições no Broker EMQX foi um pouco melhor que no Broker Mosquitto e bem superior em relação aos resultados do VerneMQ, como mostra a Tabela 6. A referida tabela apresenta o *Broker* que teve melhor Performance segundo as condições informadas.

Tabela 6 – Resultados sobre a taxa de atendimentos das requisições para o ataque *Flooding Connect*

Ataque *versus* Carga	100 requisições	500 requisições	1.000 requisições	10.000 requisições
Flooding Connect (carga: 20 bytes)	EMQX/ Mosquitto e VerneMQ	EMQX e Mosquitto	EMQX e Mosquito	EMQX

Fonte: O Autor (2024).

Continuando a análise do primeiro ataque, porém mudando a métrica para tempo de resposta, o *Broker* EMQX teve um desempenho melhor que os outros dois, como evidenciado na Tabela 7. Enquanto o VerneMQ teve o pior desempenho entre os Brokers.

Tabela 7 – Resultados relacionados ao Tempo de Resposta para o ataque *Flooding Connect*

Ataque *versus* Carga	100 requisições	500 requisições	1.000 requisições	10.000 requisições
Flooding Connect (carga: 20 bytes)	EMQX	EMQX	EMQX	EMQX

Fonte: O Autor (2024).

Em relação aos ataques *Flooding* baseado em carga útil para 5 e 20 MB, os resultados sobre a taxa de atendimento das requisições estão resumidos na Tabela 8, que evidencia qual Broker apresentou melhor desempenho. O Broker Mosquitto foi melhor que o Broker EMQX e bem superior em relação aos resultados do VerneMQ para os dois tipos de carga.

Tabela 8 – Resultados sobre a taxa de atendimentos das requisições para o ataque *Flooding payload*

Ataque *versus* *Carga*	100 requisições	500 requisições	1.000 requisições	10.000 requisições
Flooding baseado em *payload* (carga: 5MB)	EMQX / Mosquitto e VerneMQ	EMQX e Mosquitto	EMQX e Mosquito	Mosquitto
Flooding baseado em *payload* (carga: 20MB)	EMQX / Mosquitto e VerneMQ	EMQX e Mosquitto	Mosquito	Mosquitto

Fonte: O Autor (2024).

Ao se analisar o tempo de resposta, o Mosquitto, em geral, também foi melhor do que os outros Brokers, como mostra a Tabela 9. O EMQX, embora tenha alcançado um tempo de resposta melhor do que o Mosquitto em duas situações, em solicitações subsequentes ele aumentou significativamente os tempos de resposta.

Tabela 9 – Resultados relacionado ao tempo de resposta para o ataque *Flooding payload*

Ataque *versus* *Carga*	100 requisições	500 requisições	1.000 requisições	10.000 requisições
Flooding baseado em *payload* (carga: 5MB)	Mosquitto	Mosquitto	EMQX	Mosquitto
Flooding baseado em *payload* (carga: 20MB)	EMQX	Mosquitto	Mosquitto	Mosquitto

Fonte: O Autor (2024).

No último dos ataques, o *Flooding* baseado em Qualidade de serviço, como mostra a Tabela 10. Em relação a taxa de atendimento das requisições, tanto o Broker Mosquitto quanto o EMQX atenderam a todas as requisições, o que levou a ambos estarem na citada tabela.

Tabela 10 – Resultados sobre a taxa de atendimentos das requisições para o ataque *Flooding* QoS

Ataque *versus Carga*	QoS 0	QoS 1	QoS 2
Flooding baseado em QoS (carga: 20 bytes, 1.000 requisições)	EMQX e Mosquitto	EMQX e Mosquitto	EMQX e Mosquitto

Fonte: O Autor (2024).

Em relação a métrica tempo de resposta, considerando o ataque *Flooding* baseado em qualidade de Serviço, novamente o Mosquitto teve um melhor desempenho, como mostra a Tabela 11. O EMQX teve um tempo de resposta alto em comparação ao Mosquitto. Já o VerneMQ teve os maiores tempos de resposta e um uso massivo dos seus subsistemas, especialmente a memória principal.

Tabela 11 – Resultados relacionado ao tempo de resposta para o ataque *Flooding* QoS

Ataque *versus Carga*	QoS 0	QoS 1	QoS 2
Flooding baseado em QoS (carga: 20 bytes, 1.000 requisições)	Mosquitto	Mosquitto	Mosquitto

Fonte: O Autor (2024).

É possível concluir que após diversos experimentos realizados nos três Brokers, e considerando a configuração padrão de instalação deles, o Mosquitto é o Broker mais resiliente contra os ataques de negação de serviço considerados nesta avaliação. O Mosquitto em média, atendeu mais requisições, teve um tempo de resposta melhor que os outros Brokers testados e um melhor gerenciamento do seu processamento e memória na maioria dos casos. O EMQX fica em segundo lugar, sendo melhor do que o Mosquitto em apenas alguns pontos, mas não o suficiente para ser mais resiliente que o concorrente. Por sua vez, o Broker VerneMQ teve um desempenho relativamente inferior na comparação com os outros dois Brokers avaliados.

Por fim, é importante salientar que a mudança no contexto de implantação desses Brokers, ativando o método de autenticação em suas configurações, já reduziria de forma significativa esses ataques e consequentemente, melhoraria a disponibilidade dos serviços de mensageria, deixando-os mais resilientes a este tipo de ataque, já que a autenticação desempenha um papel significativo na mitigação dos ataques de negação de serviço [7].

5. Conclusões e Trabalhos Futuros

Neste capítulo, são descritas as conclusões, as contribuições, as limitações encontradas e os trabalhos futuros desta dissertação. Na Seção 5.1, são mostradas as conclusões acerca da pesquisa desenvolvida nesta dissertação. Na Seção 5.2, são apresentadas as contribuições científicas deste trabalho. Na Seção 5.3, são expostas as limitações associadas aos resultados desta pesquisa. Por fim, na Seção 5.4 são descritas e detalhadas as oportunidades de trabalhos futuros a serem realizados como complemento a esta pesquisa.

5.1. Conclusões

Os Brokers MQTT desempenham um papel relevante nos sistemas IoT, já que toda a comunicação do protocolo MQTT passa por eles. A comunicação direta entre publicadores e assinantes não é possível, pois, toda a comunicação entre eles se dá através do Broker. Portanto, é de extrema importância avaliar a resiliência destes componentes quando estão sob diferentes tipos de ataque.

Nesta dissertação, vários experimentos relacionados à diferentes tipos de ataques de negação de serviço foram realizados. O foco em ataques de negação de serviço deve-se, entre outros motivos, ao fato de que dispositivos IoT geralmente possuem recursos de hardware limitados, tornando-os mais vulneráveis a esse tipo de ataque, o que pode afetar a disponibilidade do sistema ao esgotar seus recursos. Esta dissertação representa um avanço na avaliação da segurança desses serviços de mensageria e no conhecimento do quão resilientes eles são quando estão sob ataques de negação de serviço (DoS). Três Brokers foram avaliados, e a resiliência foi medida usando métricas relacionadas a disponibilidade, como: taxa de atendimento das requisições, tempo de resposta, utilização de CPU, utilização da memória principal e utilização da memória de disco (conhecida como SWAP).

O resultado da avaliação proposta nesta dissertação possibilitou não apenas a análise em si da resiliência de cada Broker avaliado, mas também a avaliação comparativa deles em conjunto. Dada a representatividade dos Brokers: EMQX, Mosquitto e VerneMQ atualmente em sistemas IoT, é possível inclusive ponderar como anda a resiliência destes componentes de uma forma geral.

Um ponto importante a ser relembrado é que a simples ativação da autenticação na configuração dos *Brokers* MQTT aumentam a resiliência deles frente a ataques de negação de

serviço, pois apenas publicadores e assinantes cadastrados terão acesso a enviar e ler as mensagens do *Broker*.

5.2. Contribuições

A principal contribuição deste trabalho é a avaliação prática e reproduzível da resiliência de Brokers MQTT atualmente utilizados em sistemas IoT frente a ataques de negação de serviço. Esta avaliação foi conduzida utilizando uma referência na área de avaliação de desempenho, e possibilitou que resultados relevantes fossem gerados para permitir uma análise mais completa desta resiliência.

Uma outra contribuição importante deste trabalho foi a análise em si dos Brokers avaliados. Os resultados mostraram que as implementações do Mosquitto e do Broker EMQX tiveram um desempenho melhor do que o serviço de mensageria VerneMQ. O EMQX teve um melhor desempenho do que o Mosquitto em relação a taxa de atendimento de requisições para o ataque *Flooding CONNECT*, assim também aconteceu com o tempo de resposta para o mesmo tipo de ataque. Já o Broker Mosquitto teve uma taxa melhor do atendimento das requisições e um melhor tempo de resposta para todos os outros ataques e cargas. Em relação ao gerenciamento de CPU e memória, o Mosquitto teve melhor desempenho para todos os ataques. O Broker VerneMQ só conseguiu ser eficiente no atendimento das requisições quando as quantidades dessas requisições não passaram de 100 solicitações por segundo.

Outra contribuição é que o trabalho descreve como esses ataques são realizados e detalha as ferramentas utilizadas para serem usadas na avaliação. A partir disso, outros pesquisadores serão capazes de reproduzir os experimentos, favorecendo a reprodutibilidade da pesquisa. Inclusive, outros Brokers e cargas poderão ser avaliados com as informações e ferramental discutidos nesta dissertação.

5.3. Limitações

A principal limitação deste trabalho foi restringir a quantidade de Brokers MQTT a serem avaliados, considerando a falta de tempo hábil para um processo de avaliação mais criterioso com diversas métricas. Neste ponto foi feita uma escolha: ou se avaliaria mais Brokers e se reduziria as métricas, ou se aumentaria as métricas e se reduziria a quantidade de Brokers a serem testados.

Desta forma, foi escolhido se avaliar mais profundamente um conjunto menor de Brokers do que a situação oposta. Por razão similar, também não foi possível avaliar o uso de outros protocolos atualmente difundidos em IoT, como por exemplo o AMQP e o COAP.

Outra limitação do nosso estudo é que focamos apenas no estudo relacionado a disponibilidade, porém para uma avaliação de segurança mais completa seria importante avaliar outros requisitos, como a confidencialidade e como ela poderia ser garantida em Brokers MQTT.

5.4. Trabalhos Futuros

Como trabalhos futuros, estão sendo consideradas as seguintes possibilidades:

- 1) estudar as arquiteturas dos Brokers para uma melhor compreensão de toda a avaliação relacionada a resiliência, ou seja, avaliar não só os Brokers MQTT, mas também os publicadores, os assinantes e aspectos de implementação destes Brokers;
- 2) avaliar outros Brokers MQTT que não foram usados nesta dissertação, como o Nano MQ, ActiveMQ e HIVEMQ, para avaliar a porcentagem de taxa de erro para o número de solicitações feitas a cada um desses Brokers e assim determinar qual deles é o mais resiliente;
- 3) considerar o ataque de negação de serviço distribuído (DDoS) em uma futura avaliação;
- 4) considerar outros tipos de ataques de negação de serviço, como *Slow* DoS, Syn *Flooding* e ICMP *Flooding* para avaliar a resiliência dos *Brokers* em outras camadas da rede IoT.

Referências

[1] SUJAY, Lionel.Number of Internet of Things (IoT) connected devices worldwide from 2019 to 2023, with forecasts from 2022 to 2030.Statista, 2023. Disponível em: https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/ (Acessado em 21/12/2023).

[2] VANKIRK, Bob.O Relatório de Ameaças Cibernéticas SonicWall 2023 lança uma nova luz sobre as linhas de frente em constante mudança e o comportamento dos agentes deameaças.SonicWall, 2024.Disponível em: https://www.sonicwall.com/pt-br/news/o-relatorio-de-ameacas-ciberneticas-sonicwall-2023-lanca-uma-nova-luz-sobre-as-linhas-de-frente-em-constante-mudanca-e-o-comportamento-dos-agentes-de-ameacas/ (acessado 24/01/2024).

[3] Ioannis Andrea, Chrysostomos Chrysostomou, and George C. Hadjichristofi. 2015. Internet of Things: Security vulnerabilities and challenges. 2015 IEEE Symposium on Computers and Communication (ISCC) (2015), 180–187. https: //api.semanticscholar.org/CorpusID:27762246.

[4] Noon Hussein and Armstrong Nhlabatsi. 2022. Living in the Dark: MQTTBased Exploitation of IoT Security Vulnerabilities in ZigBee Networks for Smart Lighting Control. IoT 3, 4 (2022), 450–472. https://doi.org/10.3390/iot3040024.

[5] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, and Alberto Coen-Porisini. 2018. REATO: REActing TO denial of service attacks in the Internet of Things.

[6] OASIS, 2022. Disponível em:https://mqtt.org/(Acessado 21-12-2023).

[7] Chifor, Bogdan-Cosmin & Bica, Ion & Patriciu, Victor-Valeriu. (2017). Mitigating DoS attacks in publish-subscribe IoT networks. 10.1109/ECAI.2017.8166463.

[8] BARROS, Aryane et al.Constrained Application Protocol. UFRJ, 2024. Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/coap/(Acessado em 24/02/2024).

[9] AMQP,2024. Disponível em: https://www.amqp.org/(Acessado em 24/02/2024).

[10] XMPP, 2024. Disponível em: https://xmpp.org/(Acessado em 24/02/2024).

[11] A. P. Haripriya and K. Kulothungan. 2019. Secure-MQTT: an efficient fuzzy logic-based approach to detect DoS attack in MQTT protocol for internet of things. Eurasip Journal on Wireless Communications and Networking 2019, 1 (2019). https://doi.org/10.1186/s13638-019-1402-8.

[12] SZTAJNBERG, Alexandre et al. Jornadas de Atualização em Informática. Protocolos de aplicação para a internet das coisas: conceitos e aspectos práticos, Porto Alegre, ed. 37, p. 106-107, 2018. Disponível em: https://sol.sbc.org.br/livros/index.php/sbc/catalog/view/23/207/428-1. Acesso em: 22 jun. 2022.

[13] GANDHI,Darshit.RabbitMQ vs MQTT vs AMQP:A Comprehensive Comparison.Medium, 2023. Disponível em: https://medium.com/@darshit.gandhi_44389/rabbitmq-vs-mqtt-vs-amqp-a-comprehensive-comparison-1af47718d7b5 (acessado 10/02/2024).

[14] SHI,Zaiming.7 Factors to Consider When Choosing MQTT Broker 2023.EMQX,2023.Disponível em:https://www.emqx.com/en/blog/7-factors-to-consider-when-choosing-mqtt-broker-2023.(acessado 10/02/2024).

[15] FRIGON,Serge.Security Breaches in IoT Enabled Vehicles a Cause for Growing Concern.Staxbill,2024.Disponível em:https://staxbill.com/security-breaches-in-iot-enabled-vehicles/. (acessado 24/01/2024)

[16] Identity Documents & Solutions. Thales, 2024. Disponível em: https://www.thalesgroup.com/en/markets/digital-identity-and-security/government/identity (acessado 26/01/2024).

[17] Raj Jain. 1991. The Art of Computer Systems Performance Analysis: Techniques For Experimental Design, Measurement, Simulation, and Modeling, NY: Wiley.

[18] Red Hat, 2024. Disponível em: https://access.redhat.com/documentation/ptbr/red_hat_enterprise_linux/8/html/system_design_guide/getting-started-with-swap_system-design-guide (acessado 27/01/2024).

[19] Code IoT. https://codeiot.org.br/ (acessado 10 de janeiro de 2024).
[20] Dizdarevic J, Carpio F, Jukan A, Masip-Bruin X. A survey of communication protocols for Internet of Things and related challenges of fog and cloud computing integration. ACM Comput Sur. 2019;51:1-29.
[21] Kraijak S, Tuwanut P. A survey on IoT architectures, protocols, applications, security, privacy, real-world implementation and future trends. 11th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM 2015). Shanghai; 2015:1-6. https://doi.org/10.1049/cp.2015.
[22] OASIS, 2019. MQTT Version 5.0. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. (Acessado 10/01/2024).
[23] S.N. Firdous, IoT-MQTT Based Denial of Service Attack Modelling and Detection, 2020. https://ro.ecu.edu.au/theses/2303/
[24] Ribeiro D, Lins F, Nóbrega O. Avaliação de Conformidade de Requisitos de Autenticação em Gateways IoT.
[25] A. Roohi, M. Adeel, and M. Ali Shah, “DDoS in IoT: A Roadmap Towards Security & Countermeasures,” in DDoS in IoT: A Roadmap Towards Security & Countermeasures, 2019, no. September, pp. 5–7
[26] P. Sethi e S. R. Sarangi, “Internet of Things: Architectures, Protocols, and Applications”, J.Electr. Comput. Eng., vol. 2017, p. 1–25, 2017, doi: 10.1155/2017/9324035
[27] H. Ning and H. Liu, “Cyber-Physical-Social Based Security Architecture for Future Internet of Things,” Adv. Internet Things, vol. 02, no. 01, pp. 1–7, 2012, doi: 10.4236/ait.2012.21001.
[28] I. Cvitić, M. Vujić, and S. Husnjak, “Classification of security risks in the iot environment,” Ann.DAAAM Proc. Int. DAAAM Symp., vol. 2015-Janua, no. 2016, pp. 731–740, 2015, doi:10.2507/26th.daaam.proceedings.102
[29] Tariq, U.; Ahmed, I.; Bashir, A.K.; Shaukat, K. A Critical Cybersecurity Analysis and Future Research Directions for the Internet of Things: A Comprehensive Review. Sensors 2023, 23, 4117. https://doi.org/10.3390/s23084117
[30] L. Hu et al., “Cooperative Jamming for Physical Layer Security Enhancement in Internet of Things,” IEEE Internet Things J., vol. 5, no. 1, pp. 219–228, 2018, doi: 10.1109/JIOT.2017.2778185.
[31] ISACA Volunteer Member, “Cybersecurity Fundamentals Study Guide,” ISACA, 2015.
[32] M. M. Hossain, M. Fotouhi and R. Hasan, "Towards an Analysis of Security Issues, Challenges, and Open Problems in the Internet of Things," 2015 IEEE World Congress on Services, New York City, NY, 2015, pp. 21-28
[33] M. Iqbal and M. Bayoumi, “Secure End-to-End key establishment protocol for resource-constrained healthcare sensors in the context of IoT,” International Conference on High Performance Computing & Simulation (HPCS), pp. 523-530, 2016
[34] Rozan Khader and Derar Eleyan. 2021. Survey of DoS/DDoS attacks in IoT. Sustainable Engineering and Innovation 3 (01 2021), 23–29. https://doi.org/10. 37868/sei.v3i1.124
[35] Fanani, G.; Riadi, I. Analysis of Digital Evidence on Denial of Service (DoS) Attack Log Based. Bul. Ilm. Sarj. Tek. Elektro 2020,2, 70.
[36] Anthi, E.; Williams, L.; Javed, A.; Burnap, P. Hardening machine learning denial of service (DoS) defences against adversarial attacks in IoT smart home networks. Comput. Secur. 2021, 108, 102352.
[37] Alaa Alatram, Leslie F. Sikos, Mike Johnstone, Patryk Szewczyk, and James Jin Kang. 2023. DoS/DDoS-MQTT-IoT: A Dataset for Evaluating Intrusions in IoT Networks Using the MQTT Protocol. Comput. Netw. 231, C (jul 2023), 8 pages. https://doi.org/10.1016/j.comnet.2023.109809
[38] TEAM,emqx. What Is MQTT and Why Is It the Best Protocol for IoT?. EMQX, 2023. Disponpivel em: https://www.emqx.com/en/blog/what-is-the-mqtt-protocol / (Acessado em 24/02/2024).
[39] NERI, Renan et al. MQTT.UFRJ,2024.Disponível em:https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/ (Acessado 10/01/2024).
[40] OASIS, 2022. MQTT - The Standard for IoT Messaging. (Acessado 21-12-2023).

[41] OASIS open.org, 2023. MQTT Version 5.0. Disponível em:https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html.
[42] Jaidip Kotak, A. Shah, and P. Rajdev. 2019. A Comparative Analysis on Security of MQTT Brokers. 7 (5 pp.)–7 (5 pp.). https://doi.org/10.1049/cp.2019.0180
[43] HIVEMQ, 2019. Introducing the MQTT Protocol - MQTT Essentials.Disponível em https://www.hivemq.com/blog/mqtt-essentials-part-1- introducing-mqtt/ (Acessado em 10/12/2023).
[44] OASIS, 2024. MQTT Version 3.1.1. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html(acessado 11 de janeiro de 2024).
[45] Gabriel et al. MQTT. UFRJ, 2024. Disponível em : https://www.gta.ufrj.br/ensino/eel878/redes1-2023-1/trabalhos/Grupo01/ (acessado 18/01/2024)
[46] LIGHT, Roger. MQTT Retained Messages Explained.Cedalo, 2023. Disponível em:https://cedalo.com/blog/mqtt-retained-messages-explained/ (Acessado 18/01/2024).
[47] DALLINGER, Laurenz.Understanding an MQTT Packet: Ultimate Guide. Cedalo, 2023. Disponível em: https://cedalo.com/blog/mqtt-packet-guide/ (Acessado 18/01/2024)
[48] TEAM,emqx.MQTT Control Packets: A Beginner's Guide.EMQX, 2023.Disponível em:https://www.emqx.com/en/blog/introduction-to-mqtt-control-packets (Acessado 19/01/2024).
[49] Bertrand-Martinez E, Dias Feio P, Brito Nascimento Vd, Kon F e Abelém A (2020),Classification and evaluation of IoT brokers: A methodology ,Internal network management J,2020; e2115. https://doi.org/10.1002/nem.
[50] NERI, Renan et al. QoS .UFRJ,2024.Disponível em: https://www.gta.ufrj.br/ensino/eel878/redes1-2019-1/vf/mqtt/#qos (acessado 27/01/2024)
[51] TEAM,HiveMQ. What is MQTT Quality of Service (QoS) 0,1, & 2? – MQTT Essentials: Part 6. HiveMQ, 2024. Disponível em:https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels/ (Acessado 27/01/2024).
[52] Lima M, Lima R, Lins F, BEHOLDER: Um Sistema de Detecção e Prevenção Contra Intrusão em Ambientes da Internet das Coisas Baseado em Processamento de Eventos Complexos
[53] Koziolek, H.; Grüner, S.; Rückert, J. A Comparison of MQTT Brokers for Distributed IoT Edge Computing; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2020; Volume 12292.
[54] Biswajeeban Mishra, Biswaranjan Mishra, and Attila Kertész. 2021. Stress-Testing MQTT Brokers: A Comparative Analysis of Performance Measurements. Energies 14 (09 2021), 5817. https://doi.org/10.3390/en14185817.
[55] Sabrina Sicari, Alessandra Rizzardi, Daniele Miorandi, and Alberto Coen-Porisini. 2018. REATO: REActing TO denial of service attacks in the Internet of Things. Computer Networks 137 (03 2018). https://doi.org/10.1016/j.comnet.2018.03.020.
[56] Gamess, E.; Ford, T.N.; Trifas, M. Performance evaluation of a widely used implementation of the MQTT protocol with large payloads in normal operation and under a DoS attack. In Proceedings of the 2021 ACM Southeast Conference, Virtual, 15–17 April 2021; Association for Computing Machinery: New York, NY, USA, 2021; pp. 154–162.
[57] Nikhil Tripathi and Neminath Hubballi. 2021. Application Layer Denial-ofService Attacks and Defense Mechanisms: A Survey. ACM Comput. Surv. 54, 4, Article 86 (may 2021), 33 pages. https://doi.org/10.1145/3448291.
[58] Alaa Alatram, Leslie F. Sikos, Mike Johnstone, Patryk Szewczyk, and James Jin Kang. 2023. DoS/DDoS-MQTT-IoT: A Dataset for Evaluating Intrusions in IoT Networks Using the MQTT Protocol. Comput. Netw. 231, C (jul 2023), 8 pages. https://doi.org/10.1016/j.comnet.2023.109809.
[59] M. Teresa Villalba, Santiago Hernández, and Raquel Lacuesta. 2018. MQTT security: A novel fuzzing approach. Wireless Communications and Mobile Computing 2018 (02 2018). https://doi.org/10.1155/2018/8261746.
[60] M. S. Harsha, B. M. Bhavani, and K. R. Kundhavai, “Analysis of vulnerabilities in MQTT security using Shodan API and implementation of its countermeasures via authentication and ACLs,” in 2018 International Conference on Advances in Computing, Communications And

Informatics, ICACCI 2018, 2018, pp. 2244–2250.
[61] Fernández-Caramés, T.; Fraga-Lamas, P. Teaching and Learning IoT Cybersecurity and Vulnerability Assessment with Shodan through practical use cases. Sensors 2020, 20, 3048
[62] H. Sochor, F. Ferrarotti, R. Ramler, Automated security test generation for mqtt using attack patterns, in: Proceedings of the 15th International Conference on Availability, Reliability and Security, ARES '20, Association for Computing Machinery, New York, NY.
[63] Erlang, 2023. Practical functional programming for a parallel world. Disponível em: https://www.erlang.org/. (Acessado 18/12/2023).
[64] EMQX, 2023. EMQX Benefits. Disponível em: http://www. emqx.io/docs/en/v4.3/#benefits. (Acessado 18/12/2023).
[65] Octavo Labs AG The VerneMQ Company, 2023. A MQTT Broker that is scalable, enterprise ready, and open source. Disponível em: https: //vernemq.com/ (Acessado 18/12/2023)
[66] K. Somasundaram and K. Selvam, "IOT – Attacks and Challenges," Int. J. Eng. Tech. Res., vol. 8, no. 9, pp. 9–12, 2018, doi: 10.31873/ijetr.8.9.67.
[67] TEAM,EMQ.A Comprehensive Comparison of Open Source MQTT Brokers 2023. EMQX,2023.Disponível em:https://emqx.medium.com/a-comprehensive-comparison-of-open-source-mqtt-brokers-2023-e70257cc5b75 (Acessado 19/05/2023).
[68] Eclipse Foudation, 2001. Eclipse Mosquitto An open-source MQTT Broker. Disponível em: https://mosquitto.org/ (Acessado em 19/12/2023)
[69] Stepan Grabovsky, Petr Cika, Vaclav Zeman, Vlastimil Clupek, Milan Svehlak, and Jan Klime. 2018. Denial of Service Attack Generator in Apache JMeter. 2018 10th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT) (2018), 1–4. https://api.semanticscholar.org/CorpusID: 59600400
[70] JHA, Nisha; POPLI, Rashmi. Comparative Analysis of Web Applications using JMeter. International Journal of Advanced Research in Computer Science, v. 8, n. 3, 2017.
[71] MAHAJAN, Gaurav; ATTAR, Vahida; KALAMKAR, Shrida. Generation of JMeter scripts for performance testing of Moodle server. In: 2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N). IEEE, 2022. p. 2277-2281.
[72] Open Web Application Security Project OWASP. 2018. OWASP IoT Top 10. Technical Report. Disponível em: https://owasp.org/www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf. (Acessado em 02/02/2024)
[73] Collectl, 2024. Disponível em: https://collectl.sourceforge.net/ (Acessado em 02/02/2024)
[74] Montgomery, Douglas C., and George C. Runger. Applied statistics and probability for engineers. John wiley & sons, 2010
[75] RabbitMQ, 2024. Disponível em: https://www.rabbitmq.com/ (Acessado 21-12-2023).
[76] NanoMQ, 2024. Disponível em: https://nanomq.io/ (Acessado 21-12-2023).

Apêndice A

Figura 25 – Informações adicionais sobre a avaliação do Broker Mosquitto

Teste gradual de performance						Desvio Padrão		
Requisições	Broker	Tipo Ataque	Atendimento das Requisições	% Taxa de erro	%SWAP	Atendimento das Requisições	% Taxa de erro	%SWAP
100	Mosquitto	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	Mosquitto	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	Mosquitto	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
10000	Mosquitto	Inundação	97,75%	2,25%	0,00%	2,32%	2,32%	0,00%
100	Mosquitto	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	Mosquitto	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	Mosquitto	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
10000	Mosquitto	Payload (5MB)	84,56%	15,44%	0,00%	7,17%	7,17%	0,00%
100	Mosquitto	Payload (20MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	Mosquitto	Payload (20MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	Mosquitto	Payload (20MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
10000	Mosquitto	Payload (20MB)	56,87%	43,13%	0,00%	14,68%	14,68%	0,00%
1000	Mosquitto	QOS-0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	Mosquitto	QOS-1	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	Mosquitto	QOS-2	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Fonte: O Autor (2024).

Apêndice B

Figura 26 – Informações adicionais sobre a avaliação do Broker EMQX

Teste gradual de performance						Desvio Padrão		
Requisições	Broker	Tipo Ataque	Atendimento das Requisições	% Taxa de erro	%SWAP	Atendimento das Requisições	% Taxa de erro	%SWAP
100	EMQX	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	EMQX	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	EMQX	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
10000	EMQX	Inundação	99,73%	0,27%	0,00%	1,00%	1,00%	0,00%
100	EMQX	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	EMQX	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	EMQX	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
10000	EMQX	Payload (5MB)	65,76%	34,24%	2,26%	19,00%	18,85%	5,90%
100	EMQX	Payload (20MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	EMQX	Payload (20MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	EMQX	Payload (20MB)	99,42%	0,90%	0,00%	1,21%	2,04%	0,00%
10000	EMQX	Payload (20MB)	39,86%	60,48%	0,23%	14,14%	13,95%	0,51%
1000	EMQX	QOS-0	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	EMQX	QOS-1	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
1000	EMQX	QOS-2	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%

Fonte: O Autor (2024).

Apêndice C

Figura 27 – Informações adicionais sobre a avaliação do Broker VerneMQ

Teste gradual de performance						Desvio Padrão		
Requisições	Broker	Tipo Ataque	Atendimento das Requisições	% T. erro	%SWAP	Atendimento das Requisições	% Taxa de erro	%SWAP
100	VerneMQ	Inundação	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	VerneMQ	Inundação	99,05%	0,95%	17,09%	1,44%	1,44%	34,04%
1000	VerneMQ	Inundação	44,86%	55,14%	23,26%	5,05%	5,05%	36,56%
10000	VerneMQ	Inundação	3,61%	96,39%	31,96%	0,89%	0,89%	42,26%
100	VerneMQ	Payload (5MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	VerneMQ	Payload (5MB)	99,50%	0,50%	25,87%	1,11%	1,11%	21,86%
1000	VerneMQ	Payload (5MB)	45,06%	54,97%	18,06%	9,64%	9,77%	30,48%
10000	VerneMQ	Payload (5MB)	4,42%	95,56%	66,76%	1,12%	1,12%	43,81%
100	VerneMQ	Payload (20MB)	100,00%	0,00%	0,00%	0,00%	0,00%	0,00%
500	VerneMQ	Payload (20MB)	97,37%	2,63%	13,38%	3,11%	3,11%	32,06%
1000	VerneMQ	Payload (20MB)	41,10%	58,90%	4,63%	5,44%	5,44%	16,91%
10000	VerneMQ	Payload (20MB)	2,45%	97,55%	0,90%	0,89%	0,89%	4,93%
1000	VerneMQ	QOS-0	41,21%	58,79%	10,33%	9,40%	9,40%	26,02%
1000	VerneMQ	QOS-1	42,23%	57,77%	9,30%	6,76%	6,76%	24,44%
1000	VerneMQ	QOS-2	43,14%	56,86%	10,69%	7,18%	7,18%	26,94%

Fonte: O Autor (2024)