COMPARAÇÃO DE DESEMPENHO COMPUTACIONAL DE PROTOCOLOS DE COMUNICAÇÃO PARA IOT

GUSTAVO BARBOSA MACHADO

TRABALHO DE CONCLUSÃO DE CURSO EM ENGENHARIA ELÉTRICA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

Brasília, Maio de 2021

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COMPARAÇÃO DE DESEMPENHO COMPUTACIONAL DE PROTOCOLOS DE COMUNICAÇÃO PARA IOT

GUSTAVO BARBOSA MACHADO

Orientador: PROF. DR. DANIEL CHAVES CAFÉ, ENE/UNB

TRABALHO DE CONCLUSÃO DE CURSO EM ENGENHARIA ELÉTRICA

BRASÍLIA-DF, 24 DE MAIO DE 2021.

UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

COMPARAÇÃO DE DESEMPENHO COMPUTACIONAL DE PROTOCOLOS DE COMUNICAÇÃO PARA IOT

GUSTAVO BARBOSA MACHADO

TRABALHO DE CONCLUSÃO DE CURSO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM ENGENHARIA ELÉTRICA.

APROVADA POR:

_____

Prof. Dr. Daniel Chaves Café, ENE/UnB
Orientador

_____

Prof. Dr. Georges Daniel Amvame Nze, ENE/UnB
Examinador interno

_____

Prof. Dr. Luís Fernando Ramos Molinaro, ENE/UnB
Examinador interno

BRASÍLIA, 24 DE MAIO DE 2021.

FICHA CATALOGRÁFICA
GUSTAVO BARBOSA MACHADO
Comparação de desempenho computacional de protocolos de comunicação para IoT
2021xv, 38p., 201x297 mm
(ENE/FT/UnB, Graduação, Engenharia Elétrica, 2021)
Trabalho de conclusão de curso - Universidade de Brasília
Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

GUSTAVO BARBOSA MACHADO (2021) Comparação de desempenho computacional de protocolos de comunicação para IoT. Trabalho de conclusão de curso em Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 38p.

CESSÃO DE DIREITOS

AUTOR: GUSTAVO BARBOSA MACHADO
TÍTULO: COMPARAÇÃO DE DESEMPENHO COMPUTACIONAL DE PROTOCOLOS DE COMUNICAÇÃO PARA IOT
GRAU: BACHAREL ANO: 2021

_____

Gustavo Barbosa Machado
gustavo.unb1@gmail.com

Agradecimentos

Gostaria de agradecer a:

Todos meus familiares, não consanguíneos e consanguíneos, que sempre me apoiaram para seguir meus sonhos na área de tecnologia. À minha mãe e minhas avós que sempre lutaram pela minha educação e foram espelhos de determinação e bravura.

Minha namorada, Rômany Louise, que me acompanhou nesta trajetória e muitas outras em busca dos meus sonhos.

Meus grandes companheiros de vida, projetos e curso: Antônio Freire, Rafael Ramos e Henrique Lima. Meu irmão, William Dalton, o qual esteve nos bons e maus momentos de toda minha vida.

E por fim, meu orientador por estar disponível apesar dos momentos tão conturbados e singulares do ano que passamos, e por ser um exemplo de docente que em nenhum momento deixou de me incentivar.

Em memória de:
Michelângelo Machado de Oliveira;
Marcos Eunício Barbosa de Brito;

Tudo por vocês.

Resumo

Entre os protocolos em ascensão para o uso em dispositivos IoT e redes pouco robustas pode-se citar o MQTT e o CoAP. O trabalho analisou e comparou a performance computacional destes dois protocolos por meio do programa Mosquitto e da biblioteca CoAPthon com experimentações de obtenção do tempo médio de envio e o tempo total de execução. Também foram configurados a um modo de uso similar, com instanciação de servidor e cliente na mesma máquina e desconsiderando aspectos dinâmicos de rede.

Com isso, apesar do CoAP ser um protocolo de comunicação relativamente novo, há muita pesquisa e desenvolvimento de recursos adicionais para incrementá-lo e melhorar sua eficiência. Porém aspectos importantes de implementação devem ser levados em conta por projetistas, principalmente se há a necessidade do uso de segurança dos dados, pois, como obtido nos testes comparativos, o desempenho de execução do CoAPthon - uma biblioteca escrita em Python e que segue rigorosamente a padronização do protocolo - foi muito inferior ao do consolidado Mosquitto nas métricas utilizadas, principalmente se for feito o envio de pacotes maiores de mensagem por meio do block-wise.

Palavras-chave: CoAP, MQTT, DTLS, SSL, IoT.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE CÓDIGOS FONTE

Capítulo 1

Contextualização

1.1 Internet das Coisas

É notório a evolução da internet, despontamento das redes móveis(a exemplo do 5G), da economia digital e principalmente o marco da indústria 4.0, que requer que as empresas se reestruturassem em termos tecnológicos e sociais, havendo uma necessidade maior por aquisição e processamento de dados de forma quase instantânea, além de monitoramento automático constante já que estes dispositivos utilizados possuem cada vez mais independência do ser humano.

E desta própria necessidade de independência e automação se firmou a definição da internet das coisas (*IoT*), um conjunto de dispositivos e sensores inteligentes que interagem entre si em rede e ligados à internet. As mudanças vão além de apenas industriais - uma de suas principais mudanças se refere aos dispositivos comuns utilizados diariamente, sendo apenas necessário uma pequena capacidade computacional e uma rede de comunicação.

Seu percursor data de 1990 quando John Romkey criou e apresentou uma torradeira elétrica que poderia ser ligada e desligada pela internet,e nos anos seguintes recebeu diversas modificações[Mancini et al. 2018].

> A Internet das Coisas ou Internet of Things (*IoT*) desponta como uma evolução da internet e um novo paradigma tecnológico, social, cultural e digital. A Internet das Coisas revolucionará os modelos de negócios e a interação da sociedade com o meio ambiente, por meio de objetos físicos e virtuais, em que esses limites se tornam cada vez mais tênues [Lacerda and Lima-Marques 2015]

Neste sentido, junto às redes que formam a Internet das Coisas e os microcontroladores, há a necessidade cada vez mais latente do uso de sensores para otimização de processos e geração de dados. Tais sensores são capazes de captar um vasto espectro de medidas, como pode ser observado na figura 1.1, daí então um dos maiores motivos para o crescente uso de dispositivos *IoT* e seu alto valor de mercado.

A sociedade 5.0 , o quinto passo das civilizações humanas ou a sociedade informacional, define que o futuro da humanidade foque no uso das novas tecnologias disruptivas do século XXI (inteligência artificial, internet das coisas, *Big Data* e outros) para avançar as economias e resolver problemas humanos, que demandam cada vez mais o uso de equipamentos computadorizados e energia [Davies 2018]. Logo, é indispensável a adequação de todas as camadas de projetos com uso eficiente de seus meios.

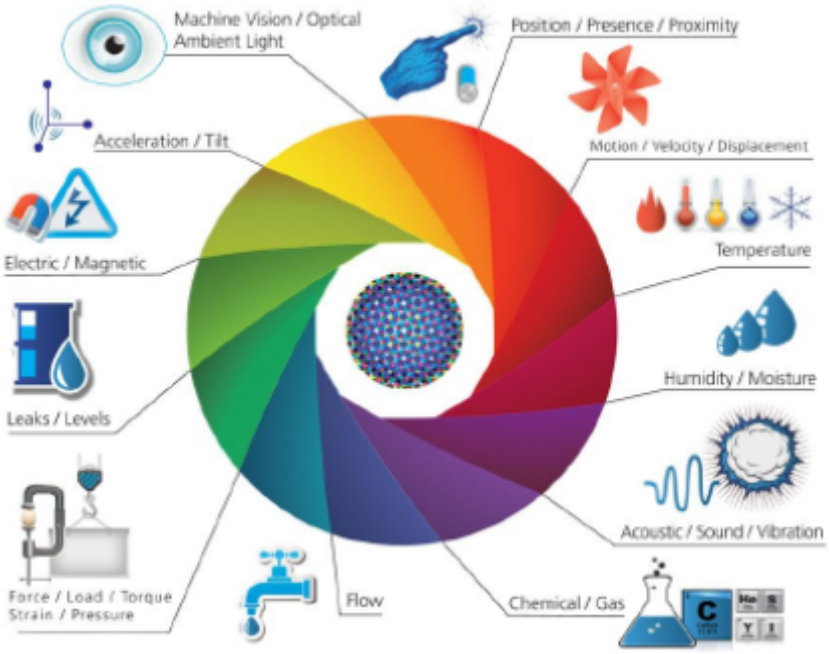


Figura 1.1: Figura retirada de Postcapes(2015)

Assim, o desenvolvimento das tecnologias de *IoT* têm potencial de mudar a economia global e melhorar a qualidade de vida dos seres humanos. Porém, apesar de toda perspectiva positiva observada pela sociedade, indústria e meio acadêmico sobre este conceito e com seu crescimento exponencial - estima-se que existam 20.4 bilhões de dispositivos ao final de 2020 [Gyarmathy 2019] -, evidencia-se a preocupação também com a segurança dos dados, a disponibilidade de serviços, autenticidade e integridade de informações.

Essa preocupação com segurança é mais notória quando percebe-se que a tecnologia penetra em áreas cruciais e sensíveis, como as já mencionadas e diversas outras áreas, como: militar, médica, infraestrutural, logística, automotiva, energia e construção. Está se tornando mais comum que as atividades comerciais e estratégicas dependam de pequenos dispositivos ligados à internet.

Os desafios de segurança se intensificam em razão das redes *IoT* estarem aumentando em quantidade e diversidade, expondo-se a diferentes tipos de ataques: *Replay*, *Man-in-Middle*, *DoS (Denial of service)*, *Flooding* e outros. Enquanto isso, há uma corrida paralela para que estes dispositivos possam executar tarefas mais numerosas e complexas, colocando em lados opostos a segurança e o desempenho [Mancini 2018].

1.1.1 Ataques

> A proteção de um recurso abstrato, como a informação, é geralmente mais difícil do que oferecer segurança física, tendo em vista ser a informação esquiva [Comer 2015].

Essa preocupação se intensifica em justificativa aos últimos ataques que foram noticiados pela mídia, os quais possuíam potencial para causar grandes estragos, sendo que muitos destes intimamente ligados a dispositivos *IoT*.

Ataques como Stuxnet [Fildes 2010] - um ataque às plantas de enriquecimento de urânio do Irã, extremamente danoso à imagem do país e um dos maiores de todos os tempos - e o vazamento de dados da NASA [Winder 2019], a agência espacial dos Estados Unidos da América, na qual a invasão foi dada por meio de um *Raspberry PI* não autorizado conectado fisicamente à rede da agência. Outro caso que teve grande repercussão foi a simulação de um ataque a um veículo [Greenberg 2015]. Nesse ataque, acadêmicos usaram um programa para invadir e controlar um carro em plena rodovia. Esse trabalho foi realizado para expor as fragilidades de sistemas veiculares interconectados.

Outro *malware* que vale a pena noticiar com mais atenção é o Mirai Botnet [Fruhlinger 2010], um ataque *DDoS (Distributed Denial of Service)* que foi usado para deixar uma parte da internet da costa leste dos Estados Unidos da América sem serviço, justamente com o uso de milhares de dispositivos *IoT*. No caso, os dispositivos foram invadidos porque usavam senhas com padrão de fábrica, e, como já divulgado, possuía uma ferramenta que analisava a rede procurando estes dispositivos específicos e evitava qualquer outros que pudessem denunciá-lo. E então estes dispositivos infectados (ou zumbificados) foram usados para congestionar os servidores de grande empresas, por meio de um ataque ilustrado na imagem 1.2, no qual o invasor tinha controle sobre estes dispositivos, e ordenava um ataque conjunto direcionado aos servidores por meio de enormes tráfegos de dados. O problema principal atualmente é que o Mirai possui diversas variações.

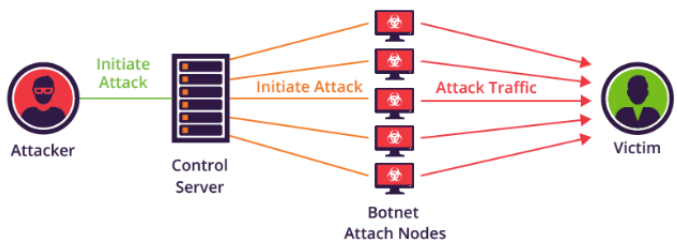


Figura 1.2: Ataque ddos - https://www.imperva.com/blog/how-to-identify-a-mirai-style-ddos-attack/

Estes casos demonstram como a excitação pela tecnologia está adiantando alguns processos essenciais antes que se perceba o risco. Um ataque mal intencionado pode causar

prejuízos financeiros, mas também em alguns casos pode ocasionar custos de vidas humanas.

Nesta vertente de proteção, com o uso de novas tecnologias de geração e manipulação de dados, no Brasil foi sancionado pelo ex-presidente Michel Temer em 2018 a LGPD, lei geral de proteção de dados pessoais N. 13709/2018, que pela definição da professora de direito digital Patrícia Peck Pinheiro no seu livro sobre a lei :

> "O espírito da lei foi proteger os direitos fundamentais de liberdade e de privacidade e o livre desenvolvimento da personalidade da pessoa natural, trazendo a premissa da boa-fé para todo o tipo de tratamento de dados pessoais, que passa a ter que cumprir uma série de princípios, de um lado, e de itens de controles técnicos, de outro lado, dentro do ciclo de vida do uso da informação que o identifique[...]"[Pinheiro 2018]

Espera-se com essa lei a discussão mais aprofundada sobre o tema e novas abordagens pelo poder legislativo e a sociedade civil, além da regulação eficiente e mais rápida para outros ramos das tecnologias.

1.2 Limitações

Logo, com um trânsito intenso de dados criados por bilhões de sistemas inteligentes embarcados e sensores (em sua maioria dados sensíveis e privados), cria-se uma preocupação com o desempenho e segurança das transações de dados.

Todavia há barreiras inerentes para a criação de ambientes seguros para *IoT*, pois suas principais características são seus pontos fracos: tratam-se de dispositivos heterogêneos, muito diversificados, e que possuem em sua totalidade restrições de hardware - pouca memória e baixa capacidade de processamento - além de se basearem em tecnologias de rede para dispositivos com baixa capacidade, como 6LoWPAN, tecnologia sem fio especificada para transmissão de pacotes IPV6 para dispositivos e redes de comunicação com pouco processamento e de baixo consumo - uma tecnologia muito importante mas falha em certos aspectos.

Em relação à comunicação, atualmente há vários protocolos de aplicação em desenvolvimento para estes sistemas. Menciona-se o MQTT, XMPP, CoAP, AMQP e outros. Cada um com suas combinações de pilha de protocolos (figura 1.3), características de segurança e de comportamento de comunicação.

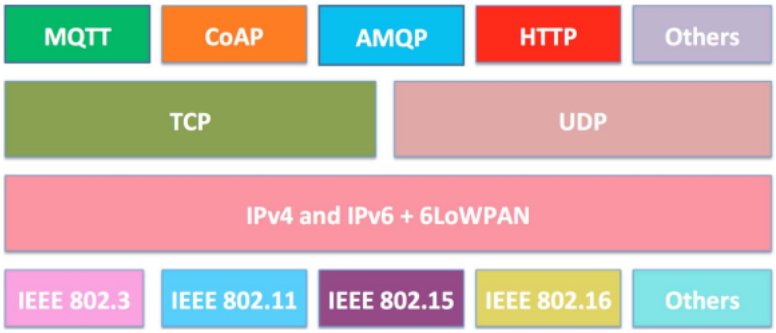


Figura 1.3: adaptado de Naik/2017

1.2.1 Objetivo geral

Diante de tamanha diversidade de protocolos disponíveis para o desenvolvimento de dispositivos *IoT*, cria-se uma complexidade para o projetista escolher o melhor protocolo de comunicação para o seu sistema. Este trabalho tem como objetivo estudar as performances de tempo de execução dos protocolos de comunicação com maior tendência de crescimento para dispositivos com baixa capacidade de processamento e memória: CoAP e MQTT. E avaliá-las em duas formas implementadas pelas comunidades de código aberto (Mosquitto e CoAPthon) em suas disposições com e sem segurança.

Capítulo 2

Revisão bibliográfica

Neste capítulo, serão apresentados conceitos importantes para o trabalho, como os de CoAP, DTLS, IPSEC e brevemente sobre as tecnologias de rede como 6LoWPAN e UDP/IP. Logo, a revisão científica dos principais trabalhos sobre o tema encontrados.

Diversos outros protocolos usados que fazem possível a comunicação via internet e que serão usados na sessão de metodologia, como mostra a pilha de protocolos TCP/IP da figura 2.1, por estarem em camadas mais baixas não serão discutidos a fundo neste trabalho.

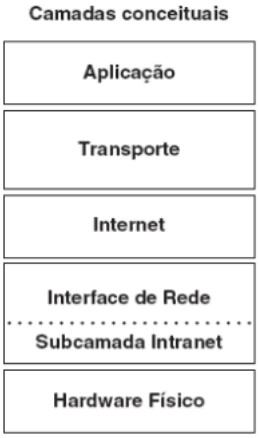


Figura 2.1: figura 4.8 (a) do livro interligação de redes com TCP/IP (Comer/2017)

2.1 6LoWPAN

IPv6 over low-power wireless personal area networks. Este protocolo transmite pacotes IPv6 através de um link IEEE 802.15.4, o que a primeira vista é inviável já que IPv6 requer um MTU de 1280 octetos e o outro tem apenas 127 a oferecer. Este protocolo encapsula e otimiza a carga de dados limitada por meio da compressão do cabeçalho dos pacotes - O padrão RFC 6282 especifica como cabeçalhos de UDP devem ser compactados do lado do transmissor e descompactados do lado do receptor.

Suporta operações requiridas no IPv6, a exemplo: descoberta de vizinho, descoberta de vizinho de forma segura (RFC 6775) e auto-configuração de endereço [Granjal et al. 2015], mas em relação a segurança, como o próprio autor pontua, não há mecanismos eficazes para esse protocolo e seu respectivo nível de camada atualmente.

2.2 UDP/IP

O protocolo de controle de transmissão (TCP) foi um dos responsáveis pela criação da internet, padronizado há 40 anos. É um protocolo da camada de transporte orientado a conexão que provê confiabilidade e ordenação dos dados transmitidos entre aplicações na internet, sendo dominante por décadas para aplicações de internet. Por ser flexível - um de seus maiores recursos de inovação - permitiu a transmissão de pacotes por redes altamente heterógeneas [Gomez et al. 2018].

O CoAP se baseia no UDP (*User Datagram Protocol*), um protocolo da camada de transporte que é encapsulada em um pacote IP para enviar e receber, sendo responsável por diferenciar as múltiplas portas de origem e destino. Porém, é mais rápido e simples que o TCP. Por estas características, difunde-se nos sistemas com restrições de recursos computacionais e de comunicação. Entretanto, uma de suas principais características é a não confiabilidade em relação a entrega do pacote, assim como não será confirmado a ordem de entrega - então, há a possibilidade de problemas com mensagens perdidas, atrasadas, duplicadas e fora de ordem que devem ser levadas em conta pelo desenvolvedor da aplicação.

Outra característica do protocolo: ele não estabelece uma conexão antes de começar a receber ou enviar. Como veremos mais à frente, o UDP se firmou como o principal protocolo para aplicações IoT, porém estudos recentes de otimizações do TCP mostram algumas situações em que sua escolha possui vantagens.

Uma das vantagens do UDP é ter o cabeçalho muito pequeno, com apenas 4 campos: porta de envio(opcional), de destino, o tamanho da mensagem e um *checksum* UDP (garante que os dados chegaram intactos). Totalizando o tamanho de 8 octetos.

2.2.1 IP - *internet protocol*

O protocolo IP é o principal protocolo de comunicação da internet, e se encontra na camada de rede. Serve para roteamento e endereçamento dos pacotes, para assim eles possam saber de onde vêm, onde querem chegar e para os roteadores saberem o caminho de retransmissão, dado que cada pacote que e enviado tem uma informação IP do dispositivo ou domínio e do destinatário. Atualmente, com a multiplicidade de endereços existentes- impulsionado inclusive pelos novos dispositivos IoT - há uma lenta transição dos coexistentes IPv4 para IPv6.

2.3 SSL/TLS

SSL (*Secure Sockets Layer*) é uma segurança baseada em criptografia e que foi transicionado para o padrão TLS (*Transport Layer Security*). TLS, como o próprio nome induz, é uma camada de segurança adicionada a camada de transporte que tem a função de segurar os serviços web. Torna-se cada vez mais rápida a transição para uso de aplicações seguras com TLS a medida que pessoas e organizações tomam conta da importância de usá-la. Principalmente com o intenso fluxo de dados sensíveis - como exemplo, pode-se mencionar o uso dominante de aplicações que usam o HTTPS sobre o HTTP.

2.4 IPSEC

IPSEC é uma abordagem de segurança diferente do SSL, pois implementa segurança diretamente na camada de internet. Tem vantagens interessantes, por exemplo a aplicação não precisa se alterar para usá-lo (isolamento da camada) e vice-versa.

IPSEC é um conjunto de protocolos criados pela IETF que garantem autenticação e privacidade na camada IP. O usuário não fica restrito ao uso de criptografias ou algoritmos de autenticação específicos, desde que seja acordado entre os pares de comunicação. Este protocolo se mostra como mais uma opção de segurança em redes inseguras.

Pode ser usado tanto com IPv4 ou IPv6. E possui dois modos de uso, o modo de transporte e de tunelamento, em que são trabalhados um sistema de autenticação ou autenticação mais criptografia dos pacotes, respectivamente.

Normalmente, não é aplicado ao CoAP e MQTT por gerar mais complexidade, mas quando possível e condizente usá-lo, dependendo da aplicação, adiciona-se mais uma camada de segurança.

2.5 DTLS

Datagram Transport Layer Security (DTLS) nada mais é do que um protocolo de segurança da camada de transporte, sendo a versão TLS baseada no UDP de segurança entre pares de comunicação.

O DTLS foi desenvolvido para redes normais, por isso há vários estudos no meio acadêmico de como fazer esta solução ficar mais amigável aos instrumentos de ambientes *IoT*, já que é uma solução primordialmente pesada em termo de consumo e processamento. Há também problemas com seu cabeçalho, que é muito longo para caber no IEEE 802.15.4 MTU - maximum transmission unit - que suporta apenas 127 bytes, mas há solução com o uso de compressão.

A preferência por seu uso se dá pelas suas características de ser flexível em relação aos serviço de segurança e criptografia. Com o CoAP, pode-se e é recomendado usar o DTLS.

Há 4 modos de uso:

- Desativado (NoSec);
- Chaves compartilhadas previamente (*PreSharedKey*);
- Par de chaves assimétricas (*RawPublicKey*);
- Certificado.

Há previamente um acordo no tipo de cifra usada no handshake para abrir uma sessão segura, podendo o pacote ficar excessivamente grande dependendo do modo de uso, por exemplo no modo com certificado.

Contempla-se o uso de ECC - *Elliptic Curve Cryptograph* - que é adotado nos modos *RawPublicKey* e certificado. Neste último há suporte ao ECDSA - *Elliptic Curve Digital Signature* - e acordo de chave com *Diffie Hellman*.

Consequentemente, *handshake* (figura 2.2) é a tarefa mais complexa do DTLS, sendo muito custoso para, por exemplo, um sensor *IoT*. Há um consenso em reduzir o tamanho do código e a quantidade de comunicação no handshake, ou como será apresentado neste trabalho,a delegação do handshake com o uso de um terceiro participante do processo de abertura da comunicação.

O *handshake* do DTLS também é o elo mais fraco em relação aos ataques DOS e DDOS, pois até clientes sem más intenções podem gerar um congestionamento com pedidos de *handshake*.

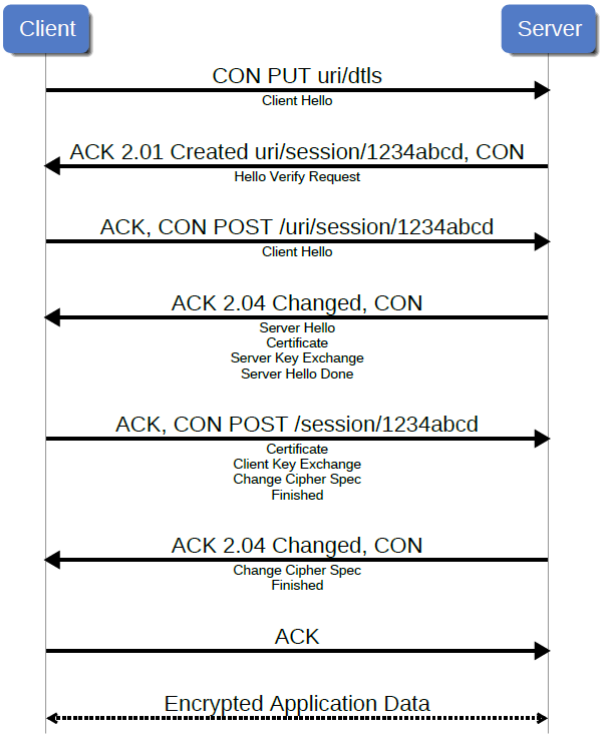


Figura 2.2: imagem do artigo security as a coap resource

2.6 HTTP

O *Hypertext Transfer Protocol* (HTTP), é um protocolo da camada de aplicação para transferência de hipertextos - informações de diversos formatos agregadas a um texto em conjunto conexo a outros textos. A importância de mencioná-lo neste trabalho se deve ao fato de que foi a base para o desenvolvimento do CoAP.

> HTTP é linguagem comum da internet moderna [Gourley 2002]

É um protocolo antigo e bem desenvolvido em que há diversas versões comerciais:

- 0.9 : Primeira versão oficial do protocolo. Pensado e usado exclusivamente para transferência de hipertextos apenas. Em outros termos, começou como um protocolo muito simples, inseguro e sem recursos primordiais usados hodiernamente devido a complexidade da rede mundial de computadores e as interações entre seus agentes.
- 1.0 : Vários avanços foram alcançados nesta versão, entre eles a criação do código de resposta de 3 dígitos, o uso de cabeçalhos e de outros tipos de mídias.
- 1.1 : Foi a versão mais longa e persiste até os dias atuais. Teve uma documentação e padronização maior - cerca de 100 mil palavras contra 700 palavras da versão 0.9 . Além disso, a criação de vários outros métodos, cookies http (possibilidade de não ser stateless) e funções como conexão persistente e cabeçalhos obrigatórios.

- 2 : versão atualmente em desenvolvimento que objetiva primordialmente melhorias na eficiência e segurança do protocolo.

Importante mencionar a consolidação do HTTP seguro ou HTTPS (uso conjunto do HTTP com uma camada de segurança extra fornecida pelo SSL/TLS), sendo unanimemente utilizado para tráfego de dados sensíveis entre clientes e servidores de empresas. O SSL/-TLS, o qual funciona entre o HTTP e o TCP, criptografa as mensagens garantindo sigilo, integridade e autenticidade entre seus pares com base em uma criptografia de chave pública. No caso, a sintaxe e o formato de mensagens são idênticos entre o HTTPS e HTTP.

Baseado por padrão nos protocolos TCP e IP, funciona no formato pedido/resposta (figura 2.3) entre aplicações e servidores, muito utilizado em navegadores web mas não se restringe a esses. Quanto às portas padrões utilizadas, são as 80, sem segurança, e 443 para tráfego seguro. Estas trocas de informações entre cliente e servidor são compostos por mensagens contendo linhas de pedido ou de *status*, cabeçalhos e carga binária.

Figura 2.3: imagem retirada do livro "HTTP - the definitive guide"

Alguns termos relacionados ao HTTP, e posteriormente ao CoAP, necessitam de uma breve explicação:

- Recursos : É a fonte de todo conteúdo web, podendo ser de qualquer tipo, inclusive

software, o qual é categorizado como recurso dinâmico. Tudo na internet é um recurso.

- URI/URL :São os nomes dos recursos e seus caminhos, sendo unicamente identificados e localizados. Assim, cada recurso tem apenas um nome. URI é o identificador do caminho e a URL o identificador do recurso em si. URLs podem direcionar para qualquer recurso da internet além dos acessados com HTTP, como : CoAP, FTP, SMTP e outros. A sua criação permitiu às aplicações a capacidade de manipular os recursos da internet de forma dinâmica e inteligente.

- Métodos: Descreve a ação realizada pelo servidor de acordo com o verbo enviado pelo cliente. O servidor então envia um código de status com a descrição da operação. Os métodos mais utilizados pode sem vistos na tabela 2.1 seguinte.

Tabela 2.1: Métodos HTTP

Métodos HTTP	Descrição
GET	Busca recurso do servidor.
PUT	Armazena recurso no servidor no local designado.
DELETE	Deleta recurso no servidor.
POST	Envia dados ao servidor de modo a criar/atualizar um recurso.

2.6.1 REST

Representational State Transfer (REST) é um estilo de arquitetura de trabalho ou formato para aplicações web descrita como o fundamento da *World Wide Web* (WWW), por meio de tecnologias como HTTP (a partir da versão 1.1), URI, JSON e outras. É comum o intercambio das palavras REST e HTTP, porém como demonstrado, essa troca é errônea porque REST é independente da tecnologia utilizada.

O desenvolvimento de aplicações baseados em REST, e o respectivo uso de serviços web, são importantes para integrar e permitir a comunicação de diversos dispositivos e softwares diferentes de forma eficiente e uniforme.

Entre as características do REST, pode-se mencionar:

- REST simplifica o uso de web service
- Adota a convenção de URIs
- É baseado em recursos
- Uso de métodos para manipular informações

- *Stateless* (sem estado), ou seja, suas ações são atômicas e integrais. Uma solicitação é independente de outras e nenhum estado será armazenado
- Tipos MIME - representam o tipo do objetos ou conteúdo: HTML, JPEG, GIF e outros. Define o tipo de entrada, saída ou retorno.

2.7 MQTT

Message Queue Telemetry Transport (MQTT) é um protocolo de mensagem M2M (máquina para máquina) da camada de aplicação destinado a conectar dispositivos por meio da internet. Protocolo de mensagem leve e de código aberto.

Idealizado para sistemas de recursos escassos (baixa largura de banda, pouco confiável ou sistemas embarcados) por mecanismos de comunicação assíncrona, assim como CoAP.

Foi idealizado pelos desenvolvedores da IBM em 1993 para operar em oleodutos, onde as redes de comunicação eram instáveis. Desde de 2013 é um protocolo padrão da *Organization for the Advancement of Structured Information Standards* (OASIS) e *open source*.

Características:
• Baseado no TCP/IP.
• Possibilidade de providenciar comunicação de um para muitos - 1:N.
• Um cabeçalho fixo de tamanho mínimo de apenas 2 bytes para mensagens de comando e o restante como opcional, como pode ser visto na figura 2.4.

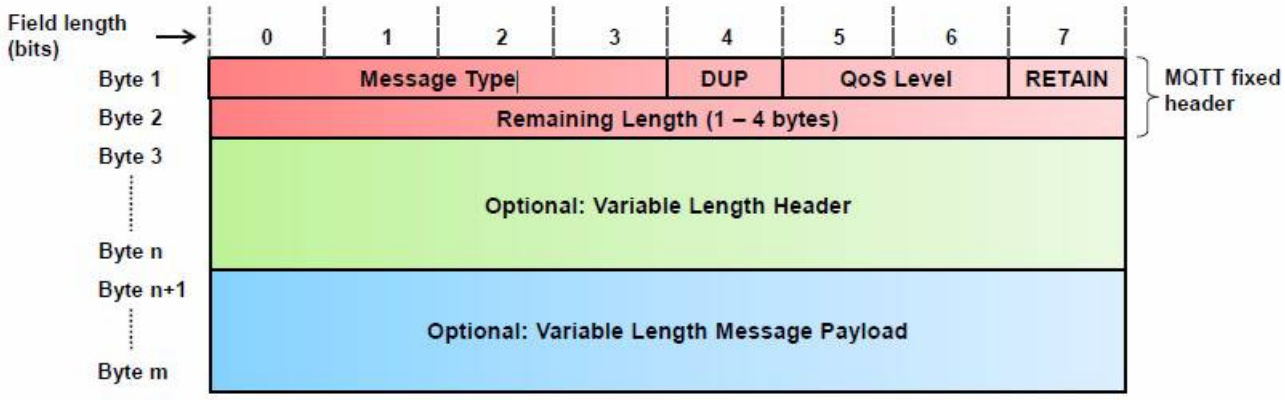


Figura 2.4: fonte: https://www.rfwireless-world.com/Tutorials/MQTT-tutorial.html

Baseia-se no sistema *publish/subscriber*, uma arquitetura que contempla 3 agentes: *publisher, subscriber e broker* - arquitetura mostrada na figura 2.5. O *broker* funciona como um servidor e elemento central. Assim, tem a função de atribuir as mensagens recebidas dos clientes aos seus endereços. Os clientes publicam e subscrevem em tópicos que são endereços nos quais as mensagem são enviadas ou recebidas. Todos os clientes subscritos em um tópico receberão as mensagens enviadas àquele tópico. O interessante dessa arquitetura é

o desacoplamento entre os agentes participantes dessa interação, por conseguinte, uma alta escalabilidade.

Similarmente ao HTTP, o MQTT funciona sobre os protocolos TCP/IP, o que lhes garante a possibilidade de confiabilidade da entrega da mensagem dependendo do modo de uso e na importância da entrega da mensagem, porém com um tamanho reduzido de mensagem.

Entre esses modos de configurações entre cliente e *broker*, temos o *Quality of Service* (QoS), definição de entrega de mensagem, e *Last Will*, a última mensagem enviada automaticamente ao desligar de um tópico. Funcionalidades as quais desempenham papéis importantes em redes instáveis.

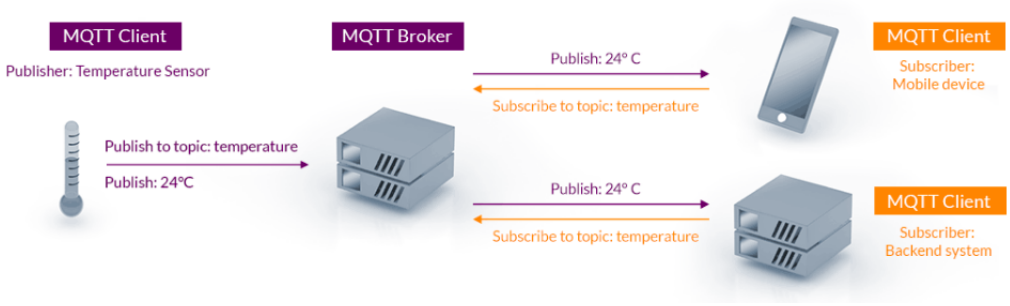


Figura 2.5: imagem retirada de mqtt.org

O QoS pode assumir os valores:

- 0 – No máximo 1 mensagem será entregue
- 1 – Envio garantido de 1 ou mais mensagens
- 2 – Envio garantido de 1 mensagem exatamente

2.8 CoAP

CoAP [Shelby et al. 2014] é um protocolo relativamente novo e especializado para transferências via web para pontos e redes restritivas em relação a recursos computacionais. Desenvolvido inicialmente para microcontroladores de 8 bits, populariza-se cada vez mais já que está sendo considerado um protocolo padrão para aplicações *IoT*. Seu uso se massificou inicialmente em aplicações na China, mas o Brasil também já apresenta uma grande quantidade de dispositivos que o utiliza [ASERT 2019].

É baseado no consistente protocolo HTTP, podendo ser considerado uma adaptação desse de forma mais leve e customizada, porém utiliza originalmente o protocolo UDP - já mencionado como assíncrono e não confiável. Utiliza também o 6LoWPAN. Foi desenhado no modelo de arquitetura REST, o que permitiu a semelhança e a retroatividade entre este e o HTTP.

Definições de uso estão na RFC (*Request for Comments*) 7252 [Shelby et al. 2014]. O

seu formato varia em alguns aspectos de acordo com seu uso, contudo, em um contexto geral segue o formato da figura 2.7. Um cabeçalho de tamanho fixo de 4 bytes seguido por um *token* de tamanho variável - entre 0 e 8 bytes - e então algumas opções e a carga binária.

CoAP segue o formato pedido/resposta, responsável pela comunicação entre cliente e servidor, possuindo seus métodos semelhantes aos HTTP: GET, PUT, POST, DELETE e outros. Todavia, trabalha também com a arquitetura *publish/subscriber* ou *resource observer*, como vista na figura 2.6 - em que é comparado as características principais entre o MQTT e CoAP.

	MQTT	CoAP
Application Layer	Single Layered completely	Single Layered with 2 conceptual sub layers (Messages Layer and Request Response Layer)
Transport Layer	Runs on TCP	Runs on UDP
Reliability Mechanism	3 Quality of Service levels	Confirmable messages, Non-confirmable messages, Acknowledgements and retransmissions
Supported Architectures	Publish-Subscribe	Request-Response, Resource observe/Publish-Subscribe

Figura 2.6: imagem retirada de Thangavel/2014

Há também uma subcamada mais baixa responsável pela troca de mensagens e que também define a qualidade de serviço, QoS, possuindo 4 tipos de mensagens que demonstram o nível de confiabilidade:

- Confirmável (CON): Requer ACK de retorno com um tempo limite preestabelecido.
- Não confirmável (NON): Não requer ACK.
- Acknowledgement (ACK): Para mensagens confirmáveis.
- Reset (RES): Para mensagens confirmáveis mas com algum contexto em falha,como problemas de processamento no destino.

Esta característica de confiabilidade e os mecanismos de retransmissão, quando utilizados, aproxima-o das características que o protocolo TCP oferece. Logo, este protocolo possui uma alta mutabilidade quanto aos seus modos de operação.

Algumas de suas ferramentas chaves são: URI e interoperabilidade com HTTP, além de diversos outros especializados como *Multicast*, *Block-Wise Transfer* (divisão de grandes pacotes em menores para envio ordenado), retransmissão e detecção de mensagens duplicadas.

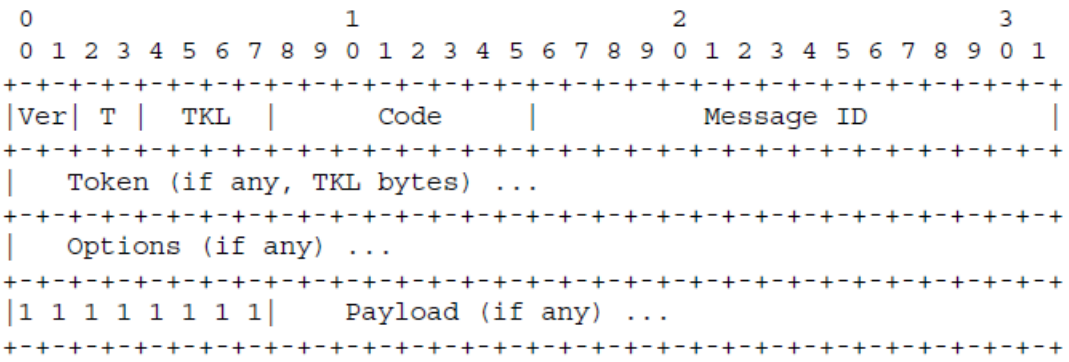


Figura 2.7: imagem retirada do RFC7252

Desde sua padronização, várias bibliotecas foram implementadas em diversas linguagens de programação, cada uma com uma especificidade e benefícios em relação às outras. Alguns exemplos podem ser vistos na figura 2.8.

Library	Version	Language	Target platform
libcoap	Develop (Sept. 24, 2016)	C	POSIX, Contiki, lwIP, TinyOS
smcp	Master (Sept. 24, 2016)	C	Embedded devices, bare-metal sensors, Linux-based devices
microcoap	Master (Sept. 24, 2016)	C	Arduino, POSIX
FreeCoAP	Master (Sept. 24, 2016)	C	GNU/Linux
Californium	1.1.0-SNAPSHOT (Sept. 24, 2016)	Java	JVM supporting devices
h5.coap	0.0.0 (Sept. 24, 2016)	JavaScript	Node.js supporting devices
node-coap	0.18.0 (Sept. 24, 2016)	JavaScript	Node.js supporting devices
CoAPthon	Master (Sept. 24, 2016)	Python	Python supporting devices
CoAPy	0.0.3-DEV (Sept. 24, 2016)	Python	Python supporting devices

Figura 2.8: imagem retirada de Iglesias/2017

2.9 Revisão científica

Nesta seção será apresentada um resumo sobre os principais trabalhos encontrados envolvendo CoAP, MQTT e comparações entre estes.

De início, foi utilizado O filtro *"IoT and Security and CoAP"* no Scopus e limitado aos anos 2014 e 2020. Os resultados obtidos são mostrados nas figuras 2.9 e 2.10.

Analisando os resultados, da imagem 2.9, podemos inferir que há um crescimento do desenvolvimento de tecnologia para a segurança do CoAP, em rápida ascensão. Já da imagem 2.10 tiramos uma notícia ruim: dos países onde mais se implementa dispositivos IoT

com CoAP (China, Brasil e EUA),não são os que mais desenvolvem material neste sentido [ASERT 2019].

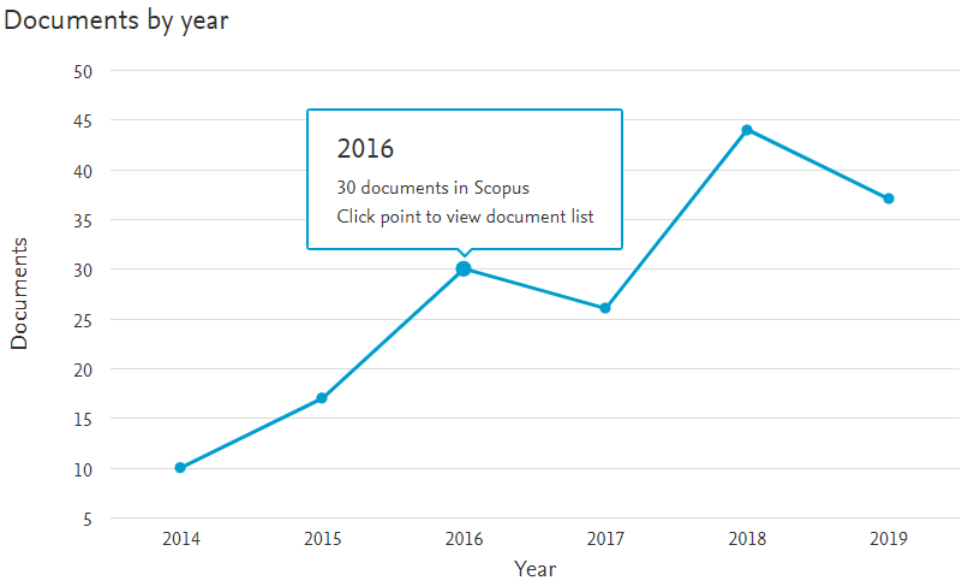


Figura 2.9: Imagem gerada pelo Scopus com filtro

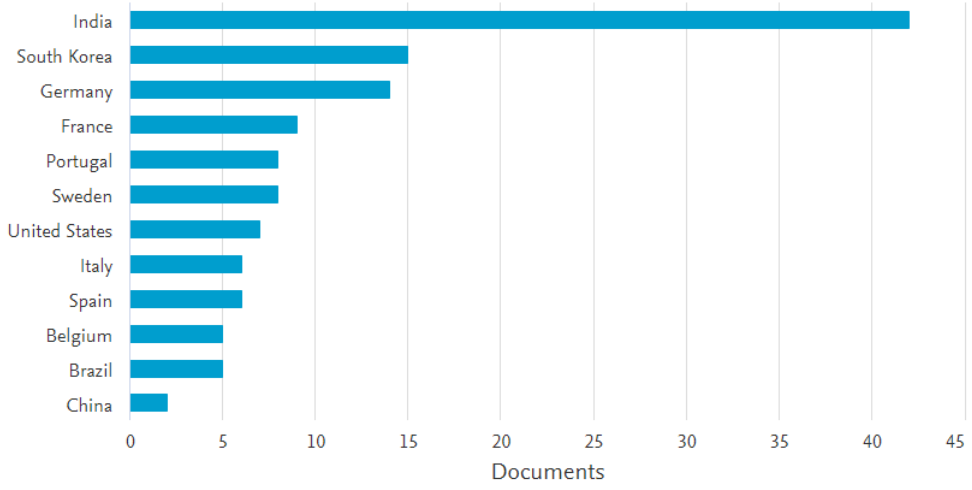


Figura 2.10: Imagem gerada pelo Scopus

Pelo Vosviewer foi gerada a imagem 2.11. Nela podemos ver um mapa de calor com relação às citações. Há alguns autores importantes, como Jorge Granjal, contudo, na maioria são pequenas ilhas de calor, ou seja, são autores que desenvolvem áreas de pesquisa heterogêneas em relação ao CoAP.

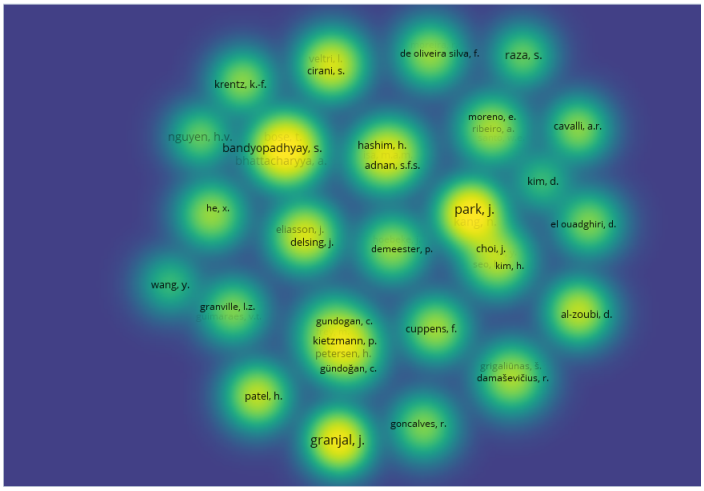


Figura 2.11: Imagem gerada pelo Vosviewer

Ressalta-se que o meio acadêmico já desenvolveu vários trabalhos em relação ao CoAP, o que evidencia o interesse crescente pelo protocolo. Como exemplo: a avaliação de desempenho empregando variações de implementações, entre eles um fino controle de acesso aos clientes [Pereira et al. 2014], e uma avaliação de desempenho das diferentes bibliotecas mais usadas e consolidadas de CoAP [Iglesias-Urkia et al. 2017]. Nesse trabalho também foram analisados e comparadas as características de cada biblioteca, como linguagens (Python, C ou Java), plataformas suportadas, segurança embarcada, extensões (*Multicast, Resource Directory* e outros) e interoperabilidade entre eles. Na figura 2.12, a seguir, mostra alguns dos resultados, sendo que foram analisadas diferentes configurações das bibliotecas em modo cliente e servidor.

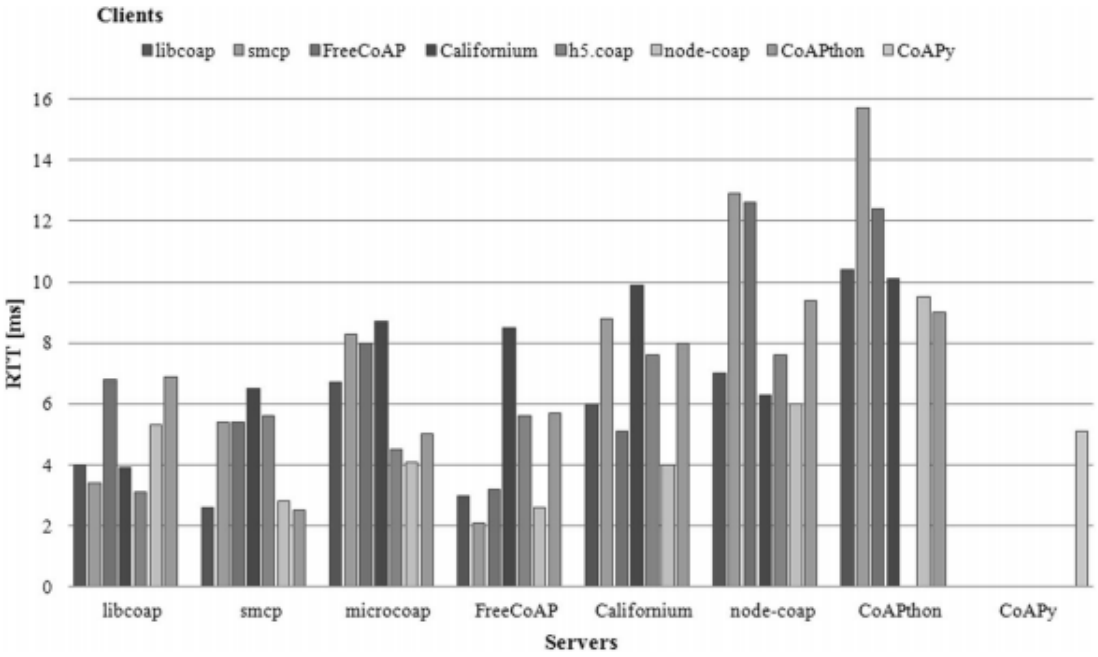


Figura 2.12: Mínimo RTT - Adaptado de iglesias 2017

2.10 Segurança

Como o CoAP trabalha sobre UDP, ele possui mecanismos para garantir confiabilidade e a ordenação das mensagens também, o que é reforçado por meio do já mencionado *Block-Wise Transfer* e as mensagens confirmáveis, que requerem o ACK [Capossele et al. 2015]. Este autor também implementa otimizações no algoritmo de ECC por meio da combinação de bibliotecas já conhecidas, que aumentaram o consumo da ROM, mas em compensação diminuíram o consumo de energia e de tempo computacional, além de utilizar um método de defesa contra ataques DOS- *Stateless Cookie Technique* - no qual os clientes, para retransmitir a abertura da comunicação, são obrigados a fazerem com o cookie acrescido, e o servidor continua o handshake apenas com a validação do cookie.

[Keoh et al. 2014] Testou o *handshake* do protocolo para diferentes reduções de bits, de espaço de memória e variações na rede, como diferentes taxas de perda de pacotes e ao final o consumo de energia.

[Raza et al. 2012] utilizou uma compressão do DTLS com uso de 6LoWPAN, reduzindo em 62% o uso de bits adicionais de segurança. Todavia, como mencionado por outros autores, não houve um estudo para saber o impacto na segurança gerado por este método.

Os trabalhos de [Granjal et al. 2015] testaram a performance do *handshake* do DTLS com ECC, além de um sistema imperioso contra ataques de repetição (*replay*) por meio do uso de valores nonce (combinação de "number"mais "once"), que são números de identificações usados uma única vez. O autor propõe ao final do trabalho que dispositivos com poucos recursos computacionais usem a delegação total do *handshake* do DTLS, usando um mecanismo baseado em TLS: *session resumption without server-side state*.

2.10.1 E-lithe

Um dos trabalhos mais promissores de [Haroon et al. 2017] foi um DTLS para IoT chamado E-lithe, uma adaptação otimizada de outro formato do DTLS chamado Lithe [Betzler et al. 2016] e que melhora consideravelmente os mecanismos contra ataques DOS, diferente do DTLS tradicional ou dos avaliados até o momento. Ele utiliza o conceito de TTP (*Trusted Third Party*) para as chaves trocadas previamente (PSK - *Pre Shared Keys*). O TTP (figura 2.13) é um mecanismo que combate ataques DoS e ajuda a poupar energia - além dos esquemas de compressão NHC (*Next Header Compression*) e IPHC (*IP Header Compression*) - estas compressões previnem a fragmentação dos pacotes, o que ajuda ainda mais na segurança. O modelo foi avaliado e comparado a outros existentes e foi melhor nos quesitos consumo, tamanho e tempo de processamento. A figura 2.13 a seguir ilustra o esquemático de funcionamento.

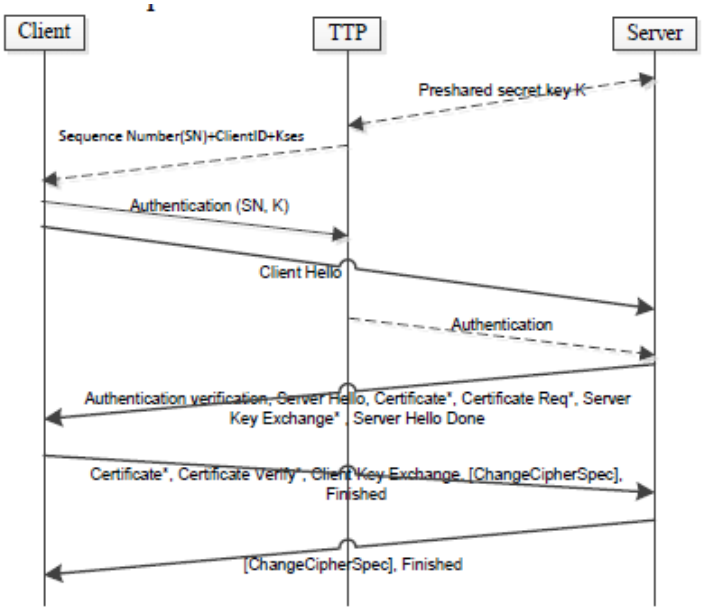


Figura 2.13: Adaptado de Haroon 2017

2.10.2 DDOS

Redes *IoT* são definidos como sendo de alta perda de pacotes (normalmente *wireless*) e baixo consumo. Isso induz que tenhamos sérios problemas com congestionamento - quando o tráfego de dados é igual ou próximo da capacidade da rede, principalmente com os ataques DOS ou DDOS (*Distributed Denial of Service*) que funciona negando os serviços por meio da exaustão de recursos do dispositivo e da rede. Tal como o já mencionado Mirae e suas variações que continuam a escanear toda a rede mundial para encontrar dispositivos vulneráveis com padrões de fábrica para serem utilizados nestes tipos de ataques. E como o CoAP está no centro dessa amplificação de uso e de vulnerabilidade, às custas das suas vantagens de ser 'zumbificado' por um intruso, precisa-se ater aos quesitos de segurança que este protocolo pode desenvolver contra ataques do tipo DDOS/DOS.

Em relação ao controle de congestionamento, o próprio CoAP possui um sistema rudimentar de controle baseado em *Exponential Backoff*. O trabalho de [Shelby et al. 2014] sugere que sejam feitos melhores sistemas de controle, já que o mecanismo é insensitivo às condições de rede. Alguns padrões como RFC 6606, comentam a importância de se desenvolver mecanismos de segurança que são adaptáveis a topologia de rede e dos dispositivos. Neste sentido, [Betzler et al. 2016] explicita o *CoCoA*, um mecanismo padronizado pela IETF para controle de congestão baseado em *Round Trip Stimation*, em que foi adaptado alguns mecanismos do TCP com mudanças devido a característica própria de alta perda de pacotes das redes, a novidade é ele que atualiza seus valores padrões a cada vez que uma nova retransmissão é efetuada - ele modifica a lógica de retransmissão com um fator de *backoff* variável. No estudo de [Järvinen et al. 2018] o CoCoA se mostrou incapaz de copiar com congestão pesada sobre caminhos de rede com *Bufferloated*.

[Ahmed and Kim 2017] Propõe o uso de redes intermediárias que analisam os pacotes

de entrada, com uma tecnologia chamada SDN (*Software-Defined Networking*), apresentada na figura 2.14. Este modelo separa o plano de controle do plano de dados, e assim esses entes derrubam pacotes/fluxos com material ou comportamento suspeito antes que chegue ao destinatário final. O formato de rede inteligente é implementado usando *of-switches* remotos por meio de um controlador central com mais recursos computacionais - que também faz amostragens de pacotes por meio de ferramentas inteligentes de análise. O problema deste formato é que redes externas não usam o formato SDN, mas se usarem é possível uma troca de informações que aumenta a eficácia do sistema proposto, já que há uma atualização constante das possíveis ameaças, além de um poder maior sobre a derrubada destes fluxos malignos. Este é um dos formatos mais promissores contra ataques DDOS, pois reduz consideravelmente ou anula seus efeitos.

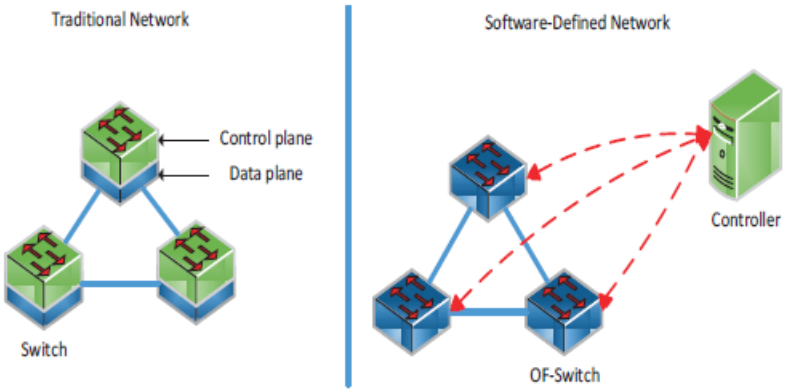


Figura 2.14: Adaptado de Ahmed/2017

2.11 CoAP sobre TCP

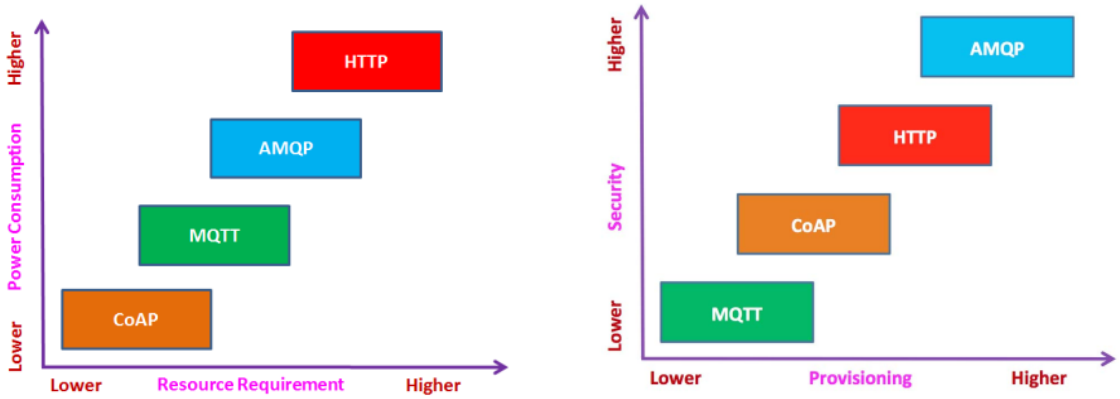
Também é importante salientar os trabalhos desenvolvidos com CoAP baseado em TCP [Bormann et al. 2018]. Criado para algumas redes que limitavam as taxas ou bloqueavam pacotes UDP. Deste modo, possui intrinsecamente um controle de congestionamento, e também tem como característica principal a entrega ordenada e controle do fluxo,oriundo do TCP.

Há uma avaliação de desempenho feita por [Jarvinen et al. 2018] deste protocolo. O trabalho mostrou que o CoAP sobre TCP é melhor que o sobre UDP - com otimizações e mecanismos de controle de congestionamento - em algumas situações específicas, como quando várias novas conexões precisam ser realizadas em situações de alto congestionamento. Apesar do CoAP sobre UDP não ter se saído mal (os avaliados foram o CoCoA e o padrão da RFC 7252), isto revela que deve-se realizar novos estudos sobre o tema já que o meio acadêmico ainda carece de pesquisa nesse sentido.

2.11.1 Comparação entre MQTT, CoAP, AMQP e HTTP

[Naik 2017] analisou de forma relativa em revisão bibliográfica os 4 protocolos de comunicação mais usados em sistemas IoT. Entre as várias comparações observadas, duas são mais evidentes: o uso dos recursos e a capacidade de segurança.

Figura 2.15: Imagem retirada de Naik/2017



O uso de recursos, segundo o autor, segue uma tendência igual da comparação do tamanho de mensagem e de cabeçalho, sendo o HTTP em posição mais desfavorável por consumir mais potência e possuir maior tamanho de mensagem devido a sua complexidade. Já a diferença encontrada pelo autor entre o CoAP e o MQTT são menores do que entre os outros protocolos - uma característica esperada.

Todavia, esses trabalhos analisados pelo autor não consideram as condições dinâmicas das redes e as condições de retransmissão. Em relação ao segundo gráfico, analisa-se que o MQTT é o protocolo mais básico em relação a segurança e recursos adicionais, algo que deve

ser levado em consideração dependendo do projeto. O autor pontua que esses protocolos podem ser combinados com outros protocolos de gerenciamento de acesso de identidade para gerar mais segurança.

2.11.2 MQTT x CoAP em meios de transmissão comuns

No trabalho de [Thangavel et al. 2014], no qual foram comparados os tempos de atraso M2M - entre o envio e o recebimento das mensagens - e o uso de banda dos protocolos MQTT e CoAP em um meio de transmissão comum, com o uso do Mosquitto e de uma biblioteca chamada Libcoap. Ademais, foram simuladas redes com taxas de perda variáveis por meio de um emulador de WAN (*Wide Area Network*). Os resultados mais importantes são demostrados nas figuras 2.16 e 2.17.

	Loss rate 0%	Loss rate 5%	Loss rate 10%	Loss rate 15%	Loss rate 20%	Loss rate 25%
MQTT (QoS1 – QoS1)	0.005289s	0.022609s	0.144252s	0.450860s	0.871025s	10.33348s
CoAP	0.008847s	0.552006s	0.642688s	1.026007s	1.886399s	2.824115s

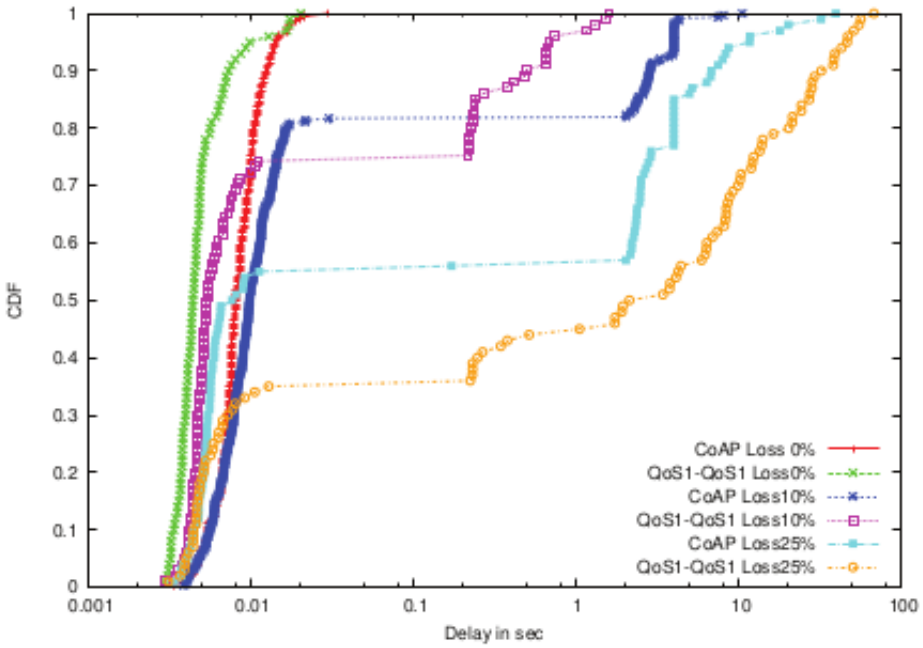


Figura 2.16: Adaptado de Thangavel 2014

Na imagem 2.16, o termo CDF significa "*Cumulative Distribution Fuction*", ou seja, é um gráfico que mostra a distribuição dos atrasos obtidos. Neste gráfico os resultados obtidos são instigantes: mostraram que o MQTT possui menor atraso para redes com perda de pacotes até 20%. O CoAP é melhor para os casos em que a perda é maior que 20%. Assim, o MQTT tem vantagem em uma longa faixa de valores. Porém, como demonstrado na figura 2.17, o consumo de banda para redes com 25% ou menos de perda de pacotes é maior que do CoAP,

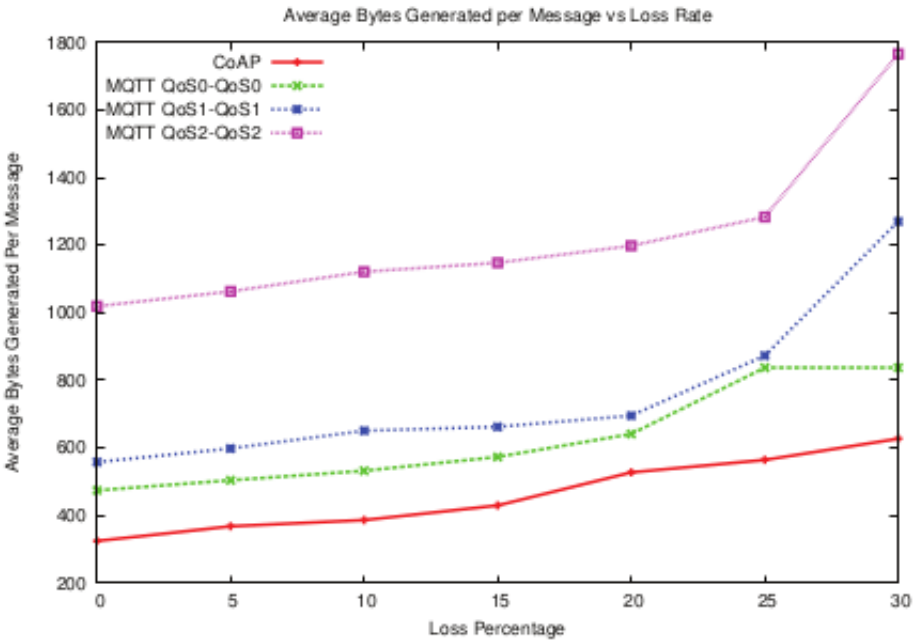


Figura 2.17: Adaptado de Thangavel 2014

tendo uma alta acentuada acima de 20%. Isso se deve a uma média maior de bytes gerados - principalmente das retransmissões - para cada mensagem enviada.

Como conclusão, o autor sugere que a escolha do protocolo por um desenvolvedor deve levar em conta o tipo de rede e suas condições intrínsecas, assim como o modo de uso.

Em suma, pode-se concluir que em relação ao CoAP há muitos trabalhos técnicos, seja em relação ao desenvolvimento da parte de software ou de arquitetura física de funcionamento, que tentam resolver problemas intrínsecos ao protocolo e assim melhorar sua eficiência e proteção.

Entretanto, há poucos trabalhos que comparam a performance dos protocolos na prática, considerando o estado atual de implementação em software disponível publicamente. Então, é proposto neste trabalho uma comparação de duas soluções de software para os protocolos MQTT e CoAP seguindo as métricas de comparação de atraso de envio; tamanho da mensagem; tempo de execução para vários envios, simulando envios de pacotes em modo blocado; e os modos criptografados.

Capítulo 3

Metodologia

Foram comparados os tempos necessários de envio de pacotes de diversos tamanhos entre o cliente e o servidor - no caso do MQTT, o cliente e o broker - e seus respectivos tempos totais de execução. Como a comparação de desempenho foi baseada nos modelos oferecidos publicamente e de uso final para usuários, as alterações no uso dos protocolos foram feitos nos arquivos de configurações e em alguns casos, no código fonte.

No trabalho, utilizou-se os *softwares* Mosquitto [Light 2017] e o CoAPthon [Tanganelli et al. 2015]. Mosquitto faz parte da *Eclipse Foundation*, uma corporação independente e sem fins lucrativos fundada pela IBM. É um software MQTT *open source* que implementa as versões 3.0 , 3.1 e 5.0. Além disso, cumpre seu papel de ser uma aplicação leve e conveniente ao uso em dispositivos IoT. Em relação à segurança, o MQTT admite o uso de SSL/TLS sobre a porta padrão 8883 TCP/IP. Sem essa segurança, a porta padrão utilizada é a 1883 TCP/IP.

O CoAPthon é uma biblioteca de CoAP escrita inteiramente em Python. E que, apesar de ser uma biblioteca recente, oferece diversas funcionalidades e está de acordo com o padrão da RFC 7252.

O material utilizado foi um *notebook* Samsung np270e5g i5 com 8 gigas de RAM e sistema operacional Ubuntu 20.04

Para gerar os certificados e chaves usados no experimento foi usado o OpenSSL, com auxílio do guia disposto em [Steve 2020], o qual foi possível gerar as chaves e certificados para criptografias RSA. A figura 3.1 demonstra um dos comandos utilizados para gerar as chaves e certificados, e a figura 3.2 demonstra estas chaves e certificados gerados.

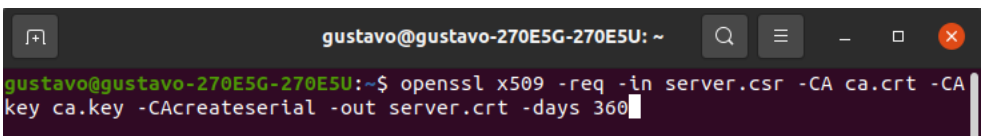


Figura 3.1: Um dos comandos para geração de chaves e certificados

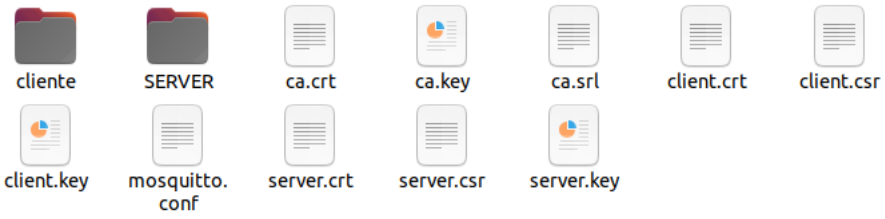


Figura 3.2: Certificados e chaves geradas por meio do OpenSSL

Destarte, foram comparados os modos sem segurança e sem outros recursos adicionais oferecidos pelos pacotes Mosquitto e CoAPthon, e o modo com segurança - SSL no Mosquitto e DTLS ao CoAPthon.

3.1 MQTT - Mosquitto

Com o Mosquitto, observou-se que a sua implementação do protocolo MQTT é sólida e bem desenvolvida. A instalação, guia de uso e as atualizações são objetivas e simples. Apesar de ter recursos adicionais mais complexos para implementação, estes são poucos em comparação ao CoAPthon.

Posto isto, foram feitos os testes com a comunicação MQTT sem SSL. A listagem 3.1 mostra o arquivo de configuração utilizado pelo Mosquitto. Os comandos usados no terminal do Linux são mostrados nas figuras 3.3 e 3.4, respectivamente a montagem do servidor e instanciação do cliente, ambos na mesma máquina.

Listagem 3.1: MQTT conf sem SSL

```
1
2 persistence true
3 persistence_location /var/lib/mosquitto/
4 listener 1883
5 allow_anonymous true
6 log_dest file /var/log/mosquitto/mosquitto.log
7
8 include_dir /etc/mosquitto/conf.d
```

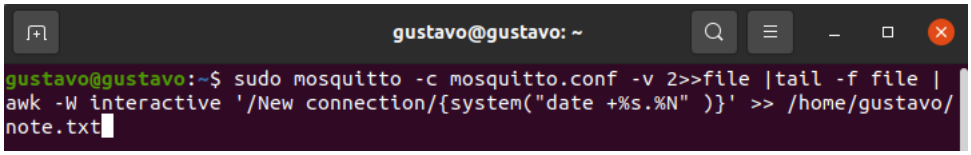


Figura 3.3: comando linux para servidor MQTT

Como mostra a figura 3.4, foi utilizado o *host* local e a porta 1883 (padrão sem criptografia), no tópico "test"e mensagem de 11 bytes. Foi utilizado QoS 2 no Mosquitto para que

Figura 3.4: comando linux para cliente MQTT

ficasse com um formato de funcionamento mais parecido com CoAP.

Nos testes realizados foi usado um marcador de tempo do Linux com precisão de nanosegundos e um comando do AWK - uma linguagem de programação procedural que procura por padrões e performa ações [Robbins 2020].

O comando AWK procura a *string* "New connection"em um arquivo de texto (com nome "file") que recebia a saída do *debugger* que era redirecionado do terminal, e então envia-se a hora de conexão com precisão de nanosegundos para outro arquivo de texto denominado "note.txt". A *string* poderia ser alterada, porque, dependendo do protocolo usado, a mensagem de aviso de recebimento é diferente.

Em outro terminal, também foi marcado a hora de execução do cliente com a mesma precisão e para o mesmo arquivo "note.txt". A diferença entre estes dois tempos foi considerada para os testes como o tempo de envio. Essa fração de tempo obtido seria o atraso entre um cliente enviar uma mensagem, após a sua completa instanciação, e o exato momento que o servidor recebe a mensagem. Ou seja, algo similar a um tempo de reação entre a execução do cliente e o sinal de recebimento do servidor, desconsiderando aspectos de rede.

Após obtidos os resultados, foram realizados os testes com SSL. A configuração utilizada é mostrada na listagem 3.2 e o comando para execução do cliente na figura 3.5. Nesta etapa foram utilizados no comando do terminal os certificados e chaves geradas pelo OpenSSL em etapas anteriores.

Listagem 3.2: conf MQTT com SSL

```
1
2  port 8883
3  cafile /etc/mosquitto/certs/ca.crt
4  certfile /etc/mosquitto/certs/server.crt
5  keyfile /etc/mosquitto/certs/server.key
6
7  tls_version tlsv1.3
8
9  allow_anonymous true
10 require_certificate true
```

Com um *script* em Python, foram automatizados os comandos no terminal do Linux para publicação das mensagens do cliente - obtidas N medidas de tempo - sendo o tempo do cliente seguido pelo tempo do servidor. Na imagem 3.6 retirada de "note.txt", é mostrado N1

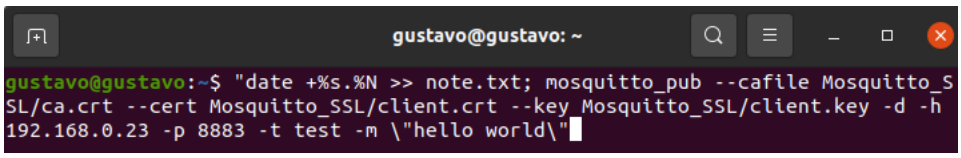


Figura 3.5: comando linux para cliente MQTT com SSL

como tempo do envio e N2 como tempo de recebimento. As medidas de tempo são iguais a N2-N1,N4-N3,N6-N5 e assim por diante.

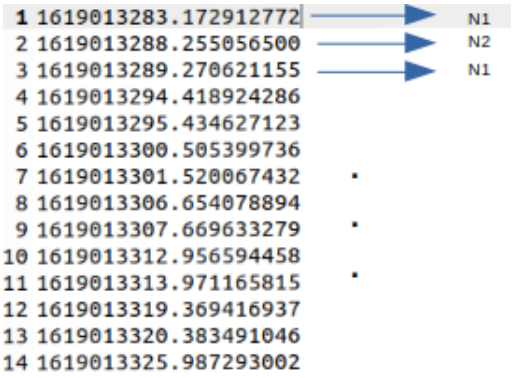


Figura 3.6: Medidas de tempo obtidas

Com a posse das medidas de tempo, foi utilizado outro *script* para calcular o tempo médio, desvio padrão e, por fim, o tempo total - o qual pode ser acessado em https://github.com/Gustavo-black-hat/TCC.git.

Em um primeiro momento, foram realizados os testes de envio para 100, 1000 e 10000 de pacotes de mensagem com a *string* de 11 bytes igual a "hello world".

Posteriormente, foi realizado outra experimentação: envio de arquivos de tamanhos variados. Os tamanho escolhidos foram de 1KB,1MB e 10MB. E foram obtidas as mesmas variáveis da primeira etapa. O comando de envio dos arquivos, sem o uso de SSL, é mostrado representativamente na imagem 3.7.

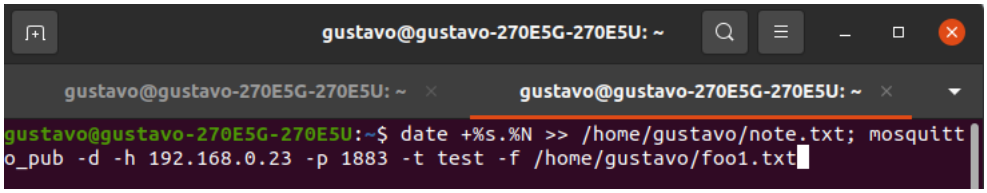


Figura 3.7: Comando para envio dos arquivos pelo mosquitto

3.2 CoAP - CoAPthon

A esquematização dos testes com CoAP foi similar ao usado no MQTT: um marcador de tempo do sistema Linux com precisão de nanosegundos e um comando do AWK para

seguir a escrita em um arquivo de texto e marcar temporalmente quando o servidor receber o sinal enviado pelo cliente. E então, com os valores de tempo, utilizou-se o *script* em Python para obter os tempos médios - e seus desvios padrões - e, por último, o tempo total de execução para enviar os N pacotes. A figura 3.11 ilustra as mensagens do depurador do servidor quando este recebe os pacotes enviados pelo cliente.

A figura 3.9 mostra o envio dos pacotes com 11 bytes para N envios iguais a 100, 1000 e 10000. A figura 3.10 mostra o comando para mil envios dos arquivos de 1KB, 1MB e 10MB.

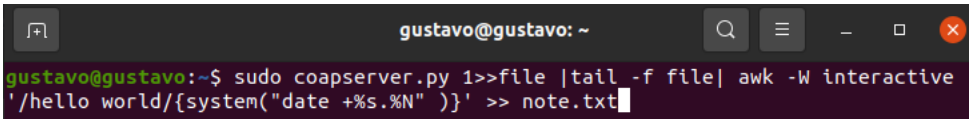


Figura 3.8: Instanciação do servidor CoAP

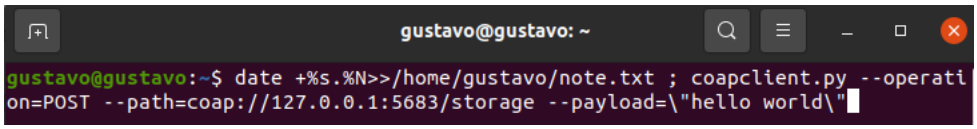


Figura 3.9: Envio da mensagem pelo cliente CoAP

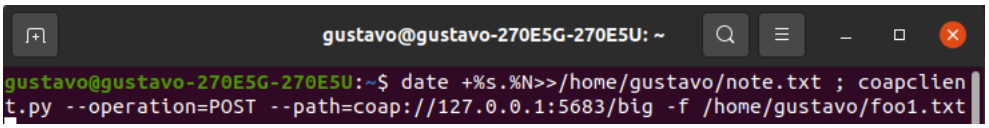


Figura 3.10: Comando para envio dos arquivos CoAP

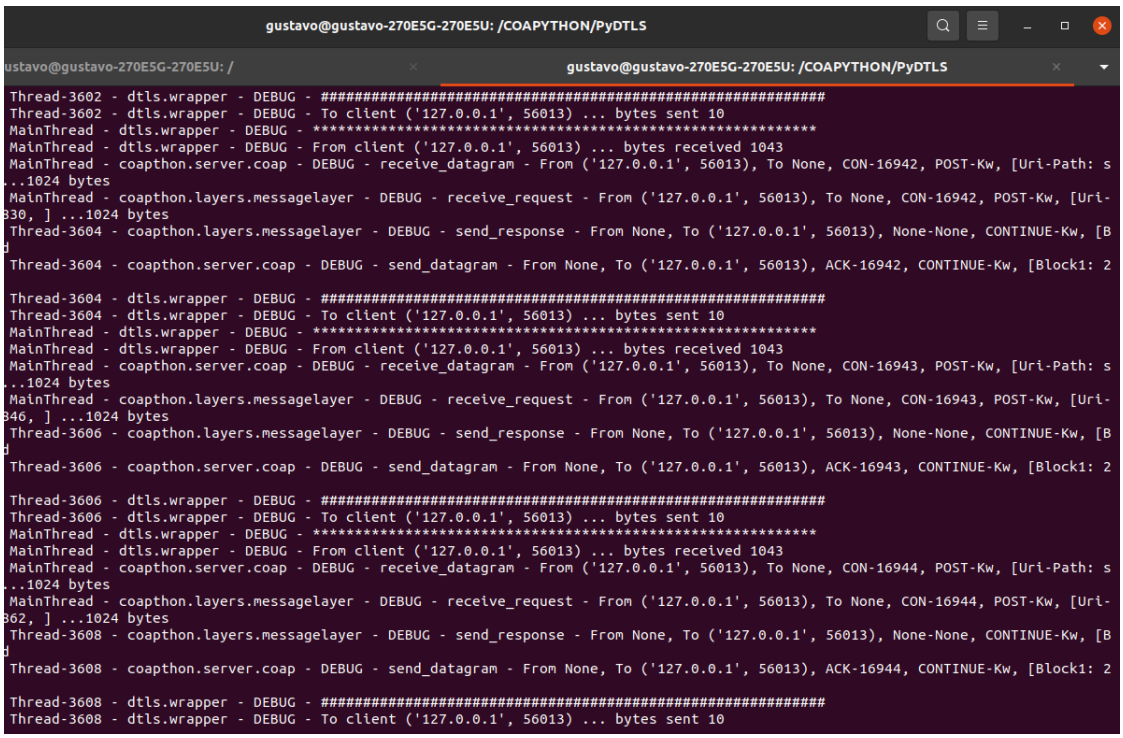


Figura 3.11: Resposta do debugger de recebimento no servidor CoAP

Para os testes do CoAP, utilizou-se como base os *scripts* em Python do CoAPthon chamados de coapclient.py e test_secure.py para gerar *scripts* próprios. Os códigos utilizados podem ser acessados por meio do link hospedados no Github em: https://github.com/Gustavo-black-hat/TCC.git.

Capítulo 4

Resultados

Neste capítulo são expostas as tabelas com os resultados obtidos para os testes propostos no capítulo anterior. Ademais, as análises destes resultados comparativos de acordo com o conhecimento adquirido ao longo da revisão.

4.1 Envio do 'hello world' em N testes

As tabelas de 4.1 a 4.4 mostram os resultados obtidos nas quatro primeiras experimentações.

Tabela 4.1: MQTT - envio do "hello world"

MQTT - Envio do "hello world"

Número de envios	Média (ms)	Desvio Padrão (ms)	Tempo de execução (s)
100	5.2	3.5	0,9
1000	3.8	1.4	6,2
10000	3.9	1.3	54,9

Tabela 4.2: MQTT - envio do "hello world"com SSL

MQTT - Envio do "hello world"com SSL

Número de envios	Média(ms)	Desvio Padrão (ms)	Tempo de execução(s)
100	5.2	3.5	1,9
1000	4.5	1.7	15,9
10000	4.6	1.9	165,3

Tabela 4.3: CoAP - envio do "hello world"

COAP - Envio do "hello world"

Número de envios	Média(ms)	Desvio Padrão(ms)	Tempo de execução(s)
100	49,7	4,8	16,0
1000	41,2	4,8	161,6
10000	42,1	4,9	1620,1

Tabela 4.4: CoAP - envio do "hello world"com DTLS

COAP - Envio do "hello world"com DTLS

Número de envios	Média(ms)	Desvio Padrão(ms)	Tempo de execução(s)
100	79,9	19,2	19,6
1000	70,3	20,3	210,6
10000	-	-	-

Os valores para 10 mil envios de "hello world"do CoAP não foram preenchidos pois o módulo DTLS do python termina em erro após alguns milhares de envios.

Em posse desses resultados, pode-se notar que um valor adequado para os outros testes são de mil envios, quando há uma estabilização dos tempos de envio de mensagens.

Analisando-se o aumento de tempo entre os envios sem e com SSL/DTLS (dados da primeira coluna), observa-se que o aumento com uso da criptografia para o Mosquitto foi de aproximadamente 20,08%. Para o CoAPthon, este aumento foi de 67,47%. Neste caso, a criptografia da biblioteca do CoAPthon teve uma performance pior do que a do Mosquitto, indicando que o tempo de execução inicial somado à iniciação do módulo de segurança deste é melhor que daquele, tanto do lado do cliente ao enviar quando do servidor ao receber.

Com relação ao tempo de execução total, os resultados indicam outra tendência. Para o MQTT, os aumentos com SSL foram respectivamente de 111,11%, 156,45% e 201,09% ante a configuração sem criptografia. Já para o CoAP, os aumentos foram de 22,2% e de 30,35%. Um aumento pequeno para o CoAP no tempo de execução total indica que o DTLS comporta-se bem para pequenos pacotes - um resultado esperado por causa do meio sem perdas de pacotes . Porém, essa comparação proporcional não indica que o DTLS funcione melhor que o SSL do Mosquitto, já que os aumentos absolutos são maiores para o DTLS: a cada mil pacotes, cerca de 10 segundos contra 50 segundos de aumento.

No geral, o CoAP foi aproximadamente 10 vezes mais lento no tempo de execução total que o Mosquitto. Pela bibliografia analisada, já esperava-se um valor superior.

4.2 Envio de arquivos com 100kB, 1MB e 10MB

Os resultados obtidos para envio dos arquivos são mostrados nas tabelas 4.6 e 4.7. O número de envios em todos os testes dessa etapa foram de mil envios - valor obtido na

experimento anterior para estabilização dos valores médios.

Tabela 4.5: MQTT - envio de arquivos com tamanhos variáveis

MQTT - Envio de arquivos de tamanhos variáveis sem SSL

Tamanhos	Média(ms)	Desvio Padrão(ms)	Tempo de execução(s)
100kB	4,2	1,7	6,1
1MB	5,4	1,8	8,2
10MB	10,8	2,18	27,7

Tabela 4.6: MQTT - Envio de arquivos com tamanhos variáveis com SSL

MQTT - Envio de arquivos com tamanhos variáveis com SSL

Tamanhos	Média(ms)	Desvio Padrão(ms)	Tempo de execução(s)
100kB	4,6	1,8	17,0
1MB	5,5	2,1	25,9
10MB	17,6	4,9	112,8

Figura 4.1: Gráfico comparativo de tempo de execução do MQTT

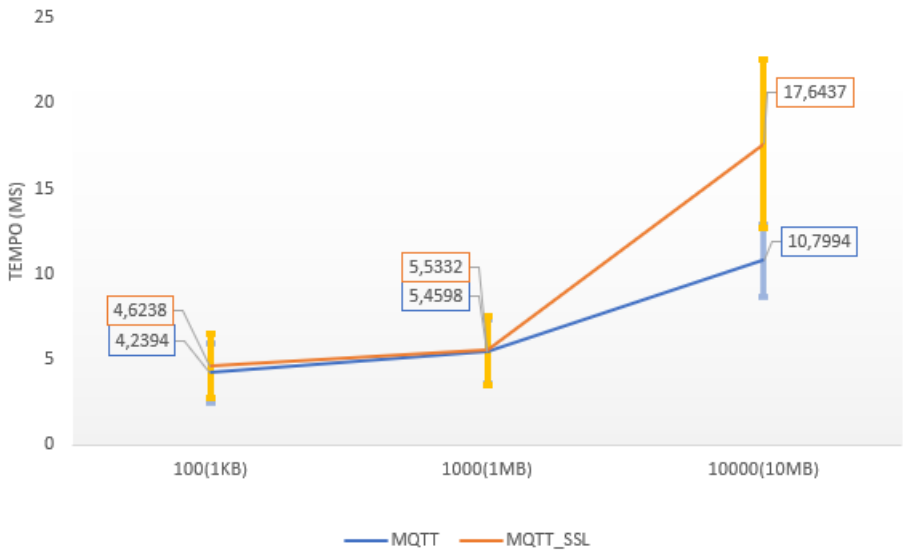


Figura 4.2: Gráfico comparativo do tempo médio de envio do MQTT

A imagem 4.1 mostra o gráfico comparativo para os valores obtidos com MQTT. Constata-se uma inclinação acentuada na direção de arquivos maiores, sendo que a diferença de tempo obtido entre os dois modos para 10MBb é 314,7% superior com SSL. Para 1MB e 100kB é de 212,6% e 176,3% respectivamente. Proporcionalmente, são valores maiores do que quando foram analisados os envios de pequenos pacotes.

O último resultado de tempo de execução total para o CoAP - tabela 4.7 - não foi preenchido com valor experimental pois o tempo estimado é exageradamente superior aos outros valores (cerca de 21 horas de experimento). O valor de média e desvio padrão foi obtido para 200 envios ao invés de mil envios. Mas com os valores obtidos é possível tirar uma conclusão do resultados e das projeções dos dados não coletados.

Tabela 4.7: CoAP - envio de arquivos

COAP - Envio de arquivos com tamanhos variáveis

Número de testes	Média(s)	Desvio Padrão(s)	Tempo de execução(s)
100 kB	0,7	0,1	1537,9
1 MB	9,6	1,1	10589,0
10 MB	120,6	8,2	98394,1*

Tabela 4.8: CoAP - envio de arquivos com DTLS

COAP - Envio de arquivos com tamanhos variáveis com DTLS

Número de testes	Média(s)	Desvio Padrão(s)	Tempo de execução(s)
100 kB	1,9	0,2	2072,5
1 MB	20,1	1,2	21169,7
10 MB	244,7	12,2	198831,3*

Os valores com "*"representam valores esperados, os quais foram calculados com base nas medidas de 200 envios.

Nota-se que o desvio padrão do CoAPthon é proporcionalmente menor que o desvio padrão do Mosquitto. E em relação ao formato das curvas obtidas (vide imagens 4.1 e 4.3), à escala logarítmica no eixo das abscissas (horizontal) fica uma impressão de um comportamento exponencial, contudo os dados indicam uma linearidade. As tabelas 4.7 e 4.8 mostram um aumento dos valores do CoAPthon progressivo com o crescimento dos pacotes, como esperado.

Os aumentos com o uso do DTLS no CoAPthon foram de 34,7%, 99% e 102%, respectivamente para os envios de 100 kB, 1 MB e 10 MB. A métrica de análise dos valores percentuais é importante porque os tempos de execução dos módulos de criptografia são dependentes dos tamanhos de mensagem.

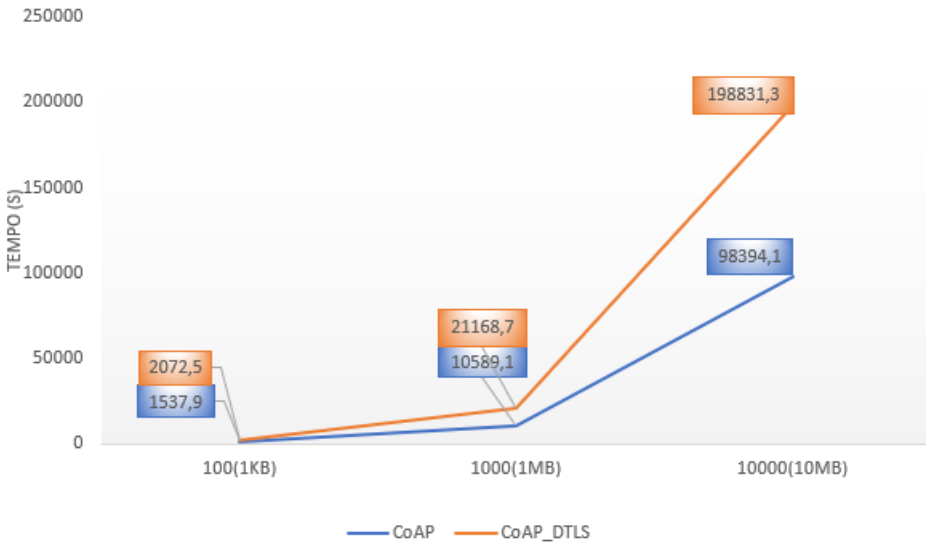


Figura 4.3: Gráfico comparativo do tempo de execução do CoAP

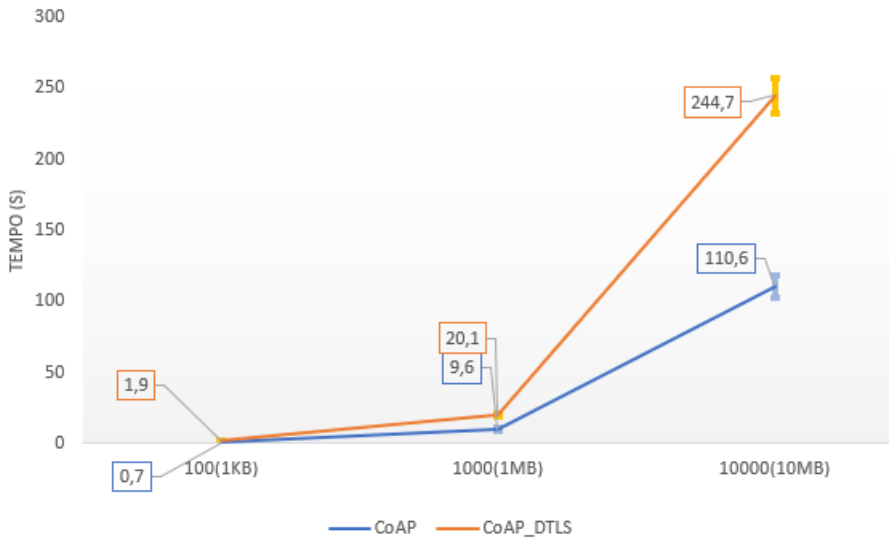


Figura 4.4: Gráfico comparativo do tempo médio de envio do CoAP

Por fim, percebe-se que o desempenho do tempo total de execução do CoAP (CoAPthon) foi muito aquém do obtido com o MQTT (Mosquitto), mesmo que não analisadas as características dinâmicas de redes e sabendo que, pelos artigos estudados nesse trabalho, o consumo de banda do CoAP é ligeiramente melhor que do MQTT devido ao tamanho reduzido de tamanho total de mensagem.

Essa diferença observada cresce com o aumento do tamanho dos arquivos. Isso se explica visto que a vantagem do CoAP - ter um cabeçalho e tamanho de mensagem total menor - reduz com a diminuição da proporção da razão cabeçalho por tamanho da mensagem. Uma das outras explicações para o pior desempenho da biblioteca do CoAP para grandes arquivos pode ser relacionado ao uso do block-wise transfer, pois cria-se mais cabeçalhos ligados às mensagens , que são transformadas em várias menores na tentativa de não perder ou duplicar pacotes. Assim, apesar de ambos os protocolos não serem especializados para grandes arquivos e terem arquiteturas similares, o CoAPthon tem uma eficiência para grandes arquivos muito menor comparado ao Mosquitto, e uma performance para pequenos pacotes com atraso desprezável dependendo da aplicação utilizada.

Em suma, os resultados obtidos foram condizentes com o esperado. Como mostra o artigo desenvolvido por [Iglesias-Urkia et al. 2017], entre as bibliotecas analisadas do CoAP, o CoAPthon foi o que teve o pior desempenho, ainda que não seja uma diferença tão grande em relação ao melhor (LibCoAP). As implementações de criptografia em ambos os protocolos comportam-se de forma similar - apesar de estarem baseados sobre protocolos de transporte diferentes (TCP e UDP). Os aumentos percentuais são relativos e devem ser observados nos casos concretos os seus verdadeiros custos.

A vantagem do MQTT utilizado pode ser explicado por vários motivos, dentre elas: o Mosquitto é escrito em C e C# - linguagens com desempenho melhor que Python em certas métricas de velocidade - e por ser designado para um simples modo de uso, seu código é menos complexo que do outro protocolo comparado. Além disso, o Mosquitto é um projeto bem estruturado e desenvolvido pela comunidade de código livre, baseado em um dos protocolos de comunicação mais antigos que existem para dispositivos com restrições e redes instáveis, o qual possui um sistema de criptografia que é bem eficiente nos padrões atuais.

Capítulo 5

Conclusão

Os esforços desenvolvidos para melhorar o desempenho dos protocolos estudados são de grande valia, visto que o número de dispositivos *IoT* tende a crescer exponencialmente principalmente com o desenvolvimento das redes 5G. Logo, com melhor eficiência nas tecnologias haverá uma economia de tempo e energia - vital para o desenvolvimento sustentável - além do aprimoramento de transferências seguras.

Estes esforços não devem ser oriundos apenas do meio científico, mas também das indústrias e empresas, pois muitas falhas de segurança e consumo exacerbado de recursos se devem a negligência e a imprudência do lançamento ao mercado de produtos no menor tempo possível, e isto se observa na análise de processos lentos e de vários casos difundidos de invasões com grandes impactos.

Quanto aos protocolos estudados, no sentido de não haver um estudo indicando uma preferência por um único, deve-se ater atenção ao CoAP. Suas especificações padronizadas no RFC são instigantes a desenvolver melhores bibliotecas já que é um protocolo muito customizável para aplicações singulares, que possuem requerimentos e naturezas únicas. Como mostrado, há diversos trabalhos para melhorar seu desempenho e também sua eficiência de segurança.

O Mosquitto apresentou resultados excelentes na metodologia usada. Mas como apresentado no artigo de [Thangavel et al. 2014], o seu uso deve ser analisado se compensatório utilizá-lo na rede se esta tiver uma alta taxa de perda de pacotes.

O CoAPthon também deve ser estudado mais a fundo para melhorar sua performance. Há pontos interessantes para o projetista que decide usa-lo: tem um desempenho eficiente para pequenos pacotes; utiliza uma linguagem fácil e intuitiva (Python); tem uma diversidade de configurações, o que permite uma escalabilidade no projeto já que ele admite os formatos *request/response* e *publish/subscriber*; e a possibilidade de compatibilização com HTTP - podendo acessar os recursos de servidores HTTP.

Para trabalhos futuros, indica-se o estudo destes protocolos com um espectro de variáveis de redes e adicionalmente o uso de analisadores de pacotes e consumo de energia.

Referências Bibliográficas

[Ahmed and Kim 2017] Ahmed, M. E. and Kim, H. (2017). Ddos attack mitigation in internet of things using software defined networking. In *2017 IEEE Third International Conference on Big Data Computing Service and Applications (BigDataService)*, pages 271–276.

[ASERT 2019] ASERT, T. (2019). Coap attacks in the wild. Disponível em: https://www.netscout.com/blog/asert/coap-attacks-wild. Acesso em: 15 nov. 2019.

[Betzler et al. 2016] Betzler, A., Gomez, C., Demirkol, I., and Paradells, J. (2016). Coap congestion control for the internet of things. *IEEE Communications Magazine*, 54(7):154–160.

[Bormann et al. 2018] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and Raymor, B. (2018). Coap (constrained application protocol) over tcp, tls, and websockets. *Internet Requests for Comments, RFC Editor, RFC*, 8323.

[Capossele et al. 2015] Capossele, A., Cervo, V., De Cicco, G., and Petrioli, C. (2015). Security as a coap resource: an optimized dtls implementation for the iot. In *2015 IEEE international conference on communications (ICC)*, pages 549–554. IEEE.

[Comer 2015] Comer, D. (2015). *Interligação de redes com TCP IP*. Number 15 in 6. CAMPUS, Rio de Janeiro.

[Davies 2018] Davies, J. (2018). Thinking ahead to society 5.0. Disponível em: https://semiengineering.com/thinking-ahead-to-society-5-0/. Acesso em: 12 fev. 2021.

[Fildes 2010] Fildes, J. (2010). Stuxnet worm 'targeted high-value iranian assets'. Disponível em: https://www.bbc.com/news/technology-11388018. Acesso em: 18 nov. 2019.

[Fruhlinger 2010] Fruhlinger, J. (2010). The mirai botnet explained: How teen scammers and cctv cameras almost brought down the internet. Disponível em: https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html. Acesso em: 18 nov. 2019.

[Gomez et al. 2018] Gomez, C., Arcia-Moret, A., and Crowcroft, J. (2018). Tcp in the internet of things: from ostracism to prominence. *IEEE Internet Computing*, 22(1):29–41.

[Gourley 2002] Gourley, D. (2002). *HTTP: The Definitive Guide*. Number 1. O'Reilly Associates, Inc, 1005 Gravenstein Highway North, Sebastopol, CA 95472.

[Granjal et al. 2015] Granjal, J., Monteiro, E., and Silva, J. S. (2015). Security for the internet of things: a survey of existing protocols and open research issues. *IEEE Communications Surveys & Tutorials*, 17(3):1294–1312.

[Greenberg 2015] Greenberg, A. (2015). Hackers remotely kill a jeep on the highway — with me in it. Disponível em: https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/. Acesso em: 10 nov. 2019.

[Gyarmathy 2019] Gyarmathy, K. (2019). Comprehensive guide to iot statistics you need to know in 2019. Disponível em: https://www.vxchnge.com/blog/iot-statistics. Acesso em: 10 dez. 2019.

[Haroon et al. 2017] Haroon, A., Akram, S., Shah, M. A., and Wahid, A. (2017). E-lithe: A lightweight secure dtls for iot. In *2017 IEEE 86th Vehicular Technology Conference (VTC-Fall)*, pages 1–5. IEEE.

[Iglesias-Urkia et al. 2017] Iglesias-Urkia, M., Orive, A., and Urbieta, A. (2017). Analysis of coap implementations for industrial internet of things: A survey. *Procedia Computer Science*, 109:188–195.

[Jarvinen et al. 2018] Jarvinen, I., Pesola, L., Raitahila, I., Cao, Z., and Kojo, M. (2018). Performance evaluation of constrained application protocol over tcp. In *2018 IEEE 88th Vehicular Technology Conference (VTC-Fall)*, pages 1–7. IEEE.

[Järvinen et al. 2018] Järvinen, I., Raitahila, I., Cao, Z., and Kojo, M. (2018). Is coap congestion safe? In *Proceedings of the Applied Networking Research Workshop*, pages 43–49. ACM.

[Keoh et al. 2014] Keoh, S. L., Kumar, S. S., and Tschofenig, H. (2014). Securing the internet of things: A standardization perspective. *IEEE Internet of things Journal*, 1(3):265–275.

[Lacerda and Lima-Marques 2015] Lacerda, F. and Lima-Marques, M. (2015). Da necessidade de princípios de arquitetura da informação para a internet das coisas. *Perspectivas em Ciência da Informação*, 20:158–171.

[Light 2017] Light, R. A. (2017). Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265.

[Mancini 2018] Mancini, M. (2018). Internet das Coisas: História, Conceitos, Aplicações e Desafios. *1*, 1(1):1.

[Naik 2017] Naik, N. (2017). Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http. In *2017 IEEE International Systems Engineering Symposium (ISSE)*, pages 1–7.

[Pereira et al. 2014] Pereira, P. P., Eliasson, J., and Delsing, J. (2014). An authentication and access control framework for coap-based internet of things. In *IECON 2014-40th Annual Conference of the IEEE Industrial Electronics Society*, pages 5293–5299. IEEE.

[Pinheiro 2018] Pinheiro, P. (2018). *Proteção de dados pessoais : Comentários à lei n. 13.709/2018 (LGPD)*. Number 15. Saraiva Jur, Rio de Janeiro.

[Raza et al. 2012] Raza, S., Trabalza, D., and Voigt, T. (2012). 6lowpan compressed dtls for coap. In *2012 IEEE 8th International Conference on Distributed Computing in Sensor Systems*, pages 287–289. IEEE.

[Robbins 2020] Robbins, A. (2020). The gnu awk user's guide. Disponível em: https://www.gnu.org/software/gawk/manual/gawk.html. Acesso em: 20 nov. 2020.

[Shelby et al. 2014] Shelby, Z., Hartke, K., Bormann, C., and Frank, B. (2014). Rfc 7252: The constrained application protocol (coap). *Internet Engineering Task Force*.

[Steve 2020] Steve (2020). Mosquitto ssl configuration -mqtt tls security. Disponível em: http://www.steves-internet-guide.com/mosquitto-tls. Acesso em: 20 nov. 2020.

[Tanganelli et al. 2015] Tanganelli, G., Vallati, C., and Mingozzi, E. (2015). Coapthon: Easy development of coap-based iot applications with python. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, pages 63–68.

[Thangavel et al. 2014] Thangavel, D., Ma, X., Valera, A., Tan, H., and Tan, C. K. (2014). Performance evaluation of mqtt and coap via a common middleware. In *2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6.

[Winder 2019] Winder, D. (2019). Confirmed: Nasa has been hacked. Disponível em: https://www.forbes.com/sites/daveywinder/2019/06/20/confirmed-nasa-has-been-hacked/?sh=77fb4b8dc627. Acesso em: 20 nov. 2019.