# Capítulo 7

# Desvendando a Camada de Aplicação na Internet das Coisas: Teoria, Prática e Tendências

Vagner E. Quincozes, Silvio E. Quincozes e Juliano F. Kazienko

***Abstract***

*The Internet of Things (IoT) is a reality that is already part of our daily lives. In this context, it is essential to understand the fundamentals of prominent protocols, especially at the application layer level, as well as aspects for implementing solutions for IoT. In this document, we introduce the application layer and its protocolos, namely MQTT, CoAP, MQTT-SN, XMPP and DDS. In addition, MQTT and CoAP protocols are presented in practice, as both have been gaining prominence in the literature and the industry, representing two alternatives for the IoT application layer. Finally, potential applications and trends in areas, namely: security, machine learning, computational paradigms, cyber-physical systems and user interfaces are addressed in order to provide insights to readers and encourage them to the development of IoT applications.*

***Resumo***

*A Internet das Coisas, do inglês, Internet of Things (IoT) é uma realidade que já faz parte do nosso cotidiano. Nesse contexto, é fundamental a compreensão dos fundamentos de protocolos proeminentes, especialmente a nível de camada de aplicação, bem como aspectos para a implementação de soluções na IoT. Neste documento, é apresentada a camada de aplicação e seus protocolos MQTT, CoAP, MQTT-SN, XMPP e DDS. Além disso, os protocolos MQTT e CoAP serão apresentados de maneira prática, visto que ambos vem ganhando destaque tanto na literatura quanto na indústria, representando duas alternativas para a camada de aplicação da IoT. Por fim, as potenciais aplicações e tendências nas áreas de segurança, aprendizado máquina, paradigmas computacionais para IoT, sistemas ciber-físicos e interfaces do usuário serão abordadas.*

## 7.1. Introdução

Avanços em áreas como sensoriamento, sistemas embarcados e microeletrônica proporcionaram o surgimento da Internet das Coisas, do inglês, *Internet of Things* (IoT). Ela pode ser compreendida como uma extensão da Internet tradicional, que inicialmente era composta de sistemas finais preponderantemente computadores de mesa, oportunizando

a conexão de objetos variados de nosso dia-a-dia a grande rede [De Farias et al. 2019] [Santos et al. 2016] [Al-Fuqaha et al. 2015] [Tanenbaum and Wetherall 2011].

A IoT provocou uma quebra de paradigma na forma como entendemos redes de computadores, tornando mais atual o conceito de uma rede formada por dispositivos [Kurose and Ross 2010]. A ideia de um mundo interligado através de objetos inteligentes, isto é, com poder de processamento, comunicação e sensoriamento, proporciona benefícios, porém apresenta desafios de ordem técnica e social [Santos et al. 2016]. Dentre os desafios destacam-se, por exemplo, os recursos computacionais dos dispositivos, que muitas vezes são limitados, tornando-se necessária a adoção de novas tecnologias ou mesmo adaptação daquelas já existentes [Quincozes et al. 2021b].

Nesse contexto, os protocolos da camada de aplicação são responsáveis por efetuar a comunicação entre tais dispositivos, fornecendo interoperabilidade na interação humano-máquina e na comunicação entre máquinas [Borgiani et al. 2021]. Existem diversos protocolos disponíveis na camada de aplicação da IoT, a saber: *Advanced Message Queuing Protocol* (AMQP), *Hyper Text Transfer Protocol Secure* (HTTP), *Message Queuing Telemetry Transport* (MQTT), MQTT *Sensor Network* (MQTT-SN), *Data Distribution Service* (DDS), *Extensible Messaging and Presence Protocol* (XMPP) e *Constrained Application Protocol* (CoAP). Dentre eles, aqueles considerados mais proeminentes e com maior adoção pela indústria são o CoAP e o MQTT [Cosmi and Mota 2019] [Quincozes et al. 2019]. De forma geral, tais protocolos destacam-se pela sua adequação a aplicações que envolvam dispositivos com recursos restritos.

Como resultado de um processo de amadurecimento de tais protocolos da camada de aplicação, tendências e cenários de pesquisa têm despontado atualmente. Dentre eles, pode-se citar o estudo de técnicas de aprendizado de máquina a fim de agregar eficiência e segurança à operação desses protocolos. Nas páginas que seguem, essa e outras tendências proeminentes são discutidas [Tahsien et al. 2020] [Aazam et al. 2018] [Humayed et al. 2017].

Neste minicurso, a camada de aplicação na IoT e seus protocolos de comunicação são apresentados. Como prática, são demonstrados casos particulares de instalação, implementação e uso envolvendo os protocolos mais promissores: o MQTT e o CoAP. Adicionalmente, os potenciais rumos em termos de pesquisa e desenvolvimento relacionados aos protocolos dessa camada são apresentados. As principais contribuições deste trabalho são:

(i) Apresentar a camada de aplicação da IoT e seus protocolos;

(ii) Demonstrar casos práticos de como instalar, implementar e utilizar, em particular, os protocolos MQTT e CoAP;

(iii) Discutir as potenciais aplicações e tendências de estudo envolvendo os protocolos da camada de aplicação relacionados à segurança da informação, aprendizado de máquina, os sistemas ciber-físicos, os paradigmas computacionais para IoT e suas interfaces do usuário.

O restante deste trabalho está organizado como segue. A Seção 7.2 aborda os fundamentos da IoT, com enfoque em sua camada de aplicação. Na Seção 7.3, os protocolos

da camada de aplicação são explanados. A Seção 7.4 apresenta os passos para instalação, implementação e utilização de casos particulares envolvendo os protocolos MQTT e CoAP. A Seção 7.5 discute as tendências relacionadas à temática tratada. Por fim, a Seção 7.6 apresenta a conclusão.

## 7.2. Fundamentos de IoT e a Camada de Aplicação

A camada de aplicação da IoT é composta por um conjunto de protocolos que comunicam dados através de *softwares* aplicativos, os quais estão distribuídos sendo executados em nós da rede. Nesta seção, são introduzidos conceitos importantes relacionados à IoT, com foco em sua camada de aplicação, a saber: evolução histórica, arquitetura de rede, sistemas operacionais, dispositivos e as arquiteturas de aplicação de rede.

### 7.2.1. Evolução Histórica

A Figura 7.1 ilustra uma linha do tempo com os principais marcos das últimas décadas que contribuíram para o desenvolvimento da IoT e seus protocolos da camada de aplicação.

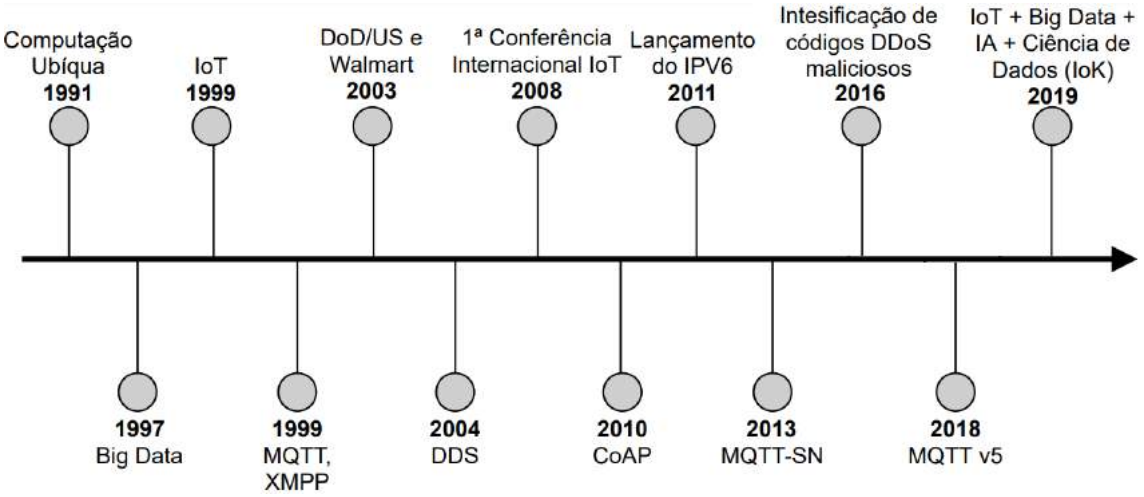



**Figura 7.1. Linha do Tempo da IoT.**

A ideia de conectar objetos começou a ser discutida na década de 1990. Mark Weiser trouxe o conceito de Computação Ubíqua em 1991 [Weiser 1991], com o objetivo de integrar a informática no cotidiano das pessoas, de forma em que os indivíduos não percebessem que estavam destinando comandos a computadores. Em 1997, surge o termo "*big data*", que visa tratar (*i.e.,* processar e armazenar) grandes quantidades de dados [Cox and Ellsworth 1997]. Mais tardar, em 1999, Kevin Ashton cunhou o termo da Internet das Coisas (IoT), com a interconexão de objetos utilizados no dia a dia à Internet, surgindo os objetos "inteligentes" [Ashton et al. 2009].

Um marco importante a ressaltar consiste na publicação do protocolo MQTT, ocorrida no ano de 1999 [OASIS 2019]. Em meados de 2000, uma primeira versão do protocolo XMPP é disponibilizada publicamente [Saint-Andre et al. 2004]. Já no ano de 2003, instituições e empresas começam a utilizar de forma intensa um método de identificação através de sinais de rádio, denominada *Radio-Frequency IDentification*

(RFID). Em 2004, a versão 1.0 do protocolo DDS é publicada [Pardo-Castellote 2003]. Em 2008, aconteceu a primeira conferência internacional designada à IoT, realizada pela Zurich [Floerkemeier et al. 2008]. A partir deste momento, surgem novas discussões e debates sobre a IoT nos meios acadêmicos. Já no ano de 2010, surge o protocolo CoAP, especializado para dispositivos restritos [Shelby et al. 2014].

Com o intenso crescimento da IoT e de seus dispositivos, a disponibilidade dos endereços de Protocolos de Internet (IPs) estava quase esgotada. Então, em 2011, surge um novo Protocolo de Internet, denominado IPv6, aumentando a disponibilidade de IPs e colaborando ainda mais para o avanço da IoT. Em 2013, surge uma nova versão do protocolo MQTT, direcionada à redes de sensores (MQTT-SN) [Stanford-Clark and Truong 2013]. Em 2016, a IoT está em alta. Contudo, governos e empresas notam a intensificação de códigos maliciosos e de ataques de negação de serviço, do inglês *Distributed Denial of Service* (DDoS), percebendo a necessidade de combate-los através da segurança da informação [Quincozes and Kazienko 2020]. Em 2018, a versão 5.0 do protocolo MQTT é disponibilizada. Por fim, em 2019 a discussão sobre a integração da IoT com Big Data, Inteligência Artificial e Ciência de Dados tem sido aprofundada.

### 7.2.2. Arquitetura IoT

Existem diversos protocolos que compõem a arquitetura da IoT, sendo que cada um deles opera em camadas específicas de acordo com os serviços ofertados em cada camada [Tanenbaum and Wetherall 2011]. Nesta seção, as camadas da arquitetura são apresentadas, destacando-se a camada de aplicação que é objeto de estudo neste documento.

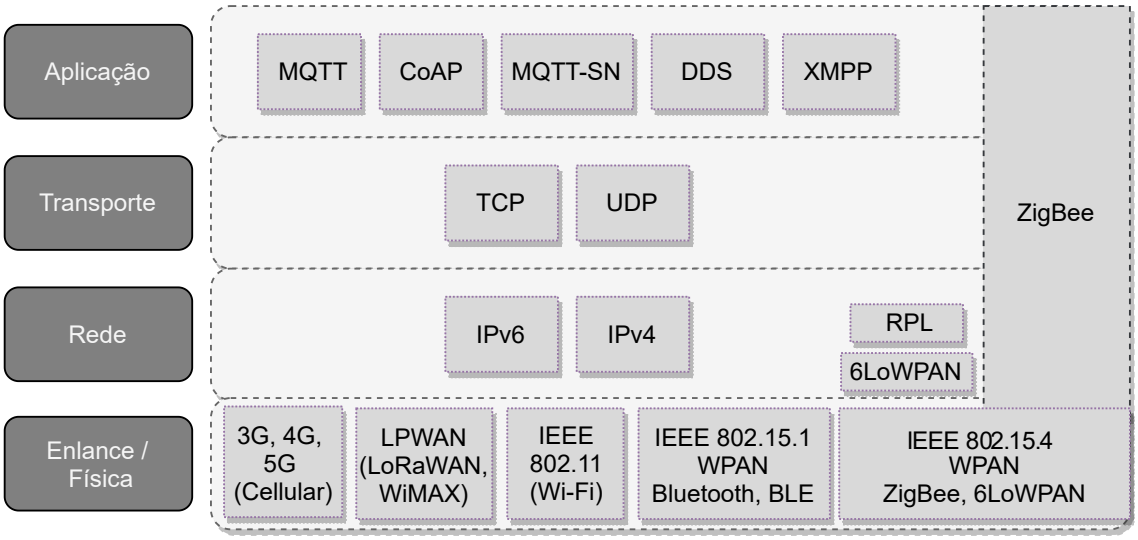



**Figura 7.2. Arquitetura IoT. Adaptado de [Quincozes et al. 2019] e [Al-Fuqaha et al. 2015].**

Conforme ilustrado na Figura 7.2, a primeira camada da arquitetura IoT é a de aplicação. Essa camada é composta por protocolos responsáveis pela comunicação entre dispositivos. Os protocolos MQTT [OASIS 2019], MQTT-SN [OASIS 2020], CoAP [Shelby et al. 2014], XMPP [Saint-Andre et al. 2004] e DDS [Pardo-Castellote 2003] são descritos e apresentados com mais detalhes na Seção 7.3.

A segunda camada da arquitetura IoT é a de transporte. A função dessa camada é realizar a transferência de dados entre máquinas. A transferência pode ser orientada a conexão ou sem conexão (datagrama). Existem dois principais protocolos nesta camada, que podem ser utilizados em aplicações IoT: o *Transmission Control Protocol* (TCP) [Postel 1981] e o *User Datagram Protocol* (UDP) [Postel 1980]. O protocolo de transporte TCP fornece confiabilidade na entrega de mensagens, pois garante que os dados transmitidos cheguem corretamente ao receptor. Já o protocolo UDP não é considerado confiável. Ele não controla o fluxo de transmissão e não retorna nenhuma mensagem confirmando que os dados foram entregues. Desse modo, não há garantia de que tais dados cheguem ao destino [Kurose and Ross 2010].

Os protocolos de Internet estão localizados na terceira camada e são responsáveis por endereçar, entregar e evitar engarrafamento na rede. Existem diferentes versões de protocolos de Internet, como o *Internet Protocol Version 4* (IPv4) e o *Internet Protocol Version 6* (IPv6). Atualmente, a versão mais recente é o IPv6, oficializado em junho de 2012. Há uma grande diferença entre tais protocolos no que diz respeito ao espaço de endereçamento. Ou seja, enquanto o IPv4 suporta $2^{32}$, o IPv6 suporta $2^{128}$ endereços IPs. O IPv6 surgiu com o intuito de substituir o IPv4 gradualmente, visto que a capacidade de IPs disponíveis estava esgotada.

Na camada de rede, duas tecnologias se destacam. Primeiramente, *Routing Protocol for Low Power and Lossy Networks* (RPL) que é um protocolo de roteamento otimizado para redes com recursos computacionais restritos, atendendo requisitos de redes com baixa potência de dados e com perdas. O protocolo é baseado na teoria de grafos acíclicos direcionados formando um *Destination-Oriented Directed Acyclic Graph* (DODAG), onde os dados são dirigidos a um nó específico chamado raiz. A segunda tecnologia consiste no padrão *IPv6 over Low-Power Wireless Area Network* (6LoWPAN), que permite a comunicação de dados de forma eficiente em redes de baixa potência e velocidade. Ele atua adaptando o IPv6 para seu uso em redes que demandam baixo consumo IPv6 [Santos et al. 2016].

As últimas duas camadas da arquitetura, de enlace e física, podem ser definidas por várias tecnologias. Uma delas consiste no padrão IEEE 802.15.4, que permite controlar o acesso em redes sem fio pessoais com baixas taxas de transmissão [Quincozes et al. 2019] [Al-Fuqaha et al. 2015]. Outra tecnologia bastante adotada nesta camada é o *Long Range Wide Area Network* (LoRaWAN). Ela foi projetada para operar em redes de alto alcance e baixa potência, além de atender requisitos de baixo consumo energético. Na elaboração de uma rede, a tecnologia LoRaWAN implementa protocolos que definem a comunicação entre os componentes da arquitetura [Alliance 2017]. Adicionalmente, uma especificação importante para as camadas superiores consiste no ZigBee. O ZigBee tem como foco dispositivos de baixa potência e sua especificação prevê protocolos específicos executados nas camadas de rede e de aplicação [Alliance 2015].

### 7.2.3. Sistemas Operacionais e Dispositivos da IoT

Os sistemas operacionais (SO) da IoT são projetados para atender aos requisitos computacionais (capacidade de memória, energia, potência e armazenamento) dos dispositivos da IoT. Um dos sistemas operacionais mais utilizado por pessoas de todo o mundo é o Android, comumente adotado em *smartphones*. Também, ressalta-se a existência de

sistemas operacionais específicos para redes de sensores sem fio (RIOT) e microcontroladores (NuttX). Outros SO que serão apresentados consistem no Raspbian, TinyOS e Contiki, todos otimizados para operar em dispositivos com recursos restritos e atender os requisitos da IoT [Bansal and Kumar 2020][Santos et al. 2016][Hahm et al. 2015].

**Android**[1]**.** É um SO baseado no núcleo Linux, revelado em 2007 pela Google junto com a fundação *Open Handset Alliance*, com o objetivo de desenvolver a indústria de dispositivos móveis [Alliance 2007]. O Android é o SO mais utilizado na atualidade. Esse SO pode ser instalado em *smartphones*, por exemplo, permitindo que os usuários controlem outros equipamentos da IoT. O *smartphone* funciona como um controle remoto [Santos et al. 2016].

**RIOT**. É um sistema operacional de código aberto, desenvolvido por uma comunidade de pessoas distribuídas pelo mundo todo[2]. O RIOT fornece suporte à maioria dos dispositivos IoT, considerando baixa potência, microcontroladores (8, 16 e 32 bits) e demais dispositivos externos. As características do RIOT consistem no fácil desenvolvimento, suporte à segurança da camada de transporte (*e.g.,* DTLS), qualidade (*i.e.,* testes constantes) e com suporte às tecnologias da IoT, como os protocolos 6LoWPAN, MQTT-SN e CoAP[3]. Ademais, a RIOT conta com um ecossistema ciber-físico para monitorar e controlar objetos inteligentes [Bansal and Kumar 2020].

**NuttX**[4]**.** O NuttX é um sistema operacional que opera em tempo real. Ele pode ser construído como um microkernel ou como uma versão monolítica. É modular e escalável a ambientes de microcontroladores de 8 a 64 bits. Recentemente o NuttX começou a suportar a 6LoWPAN, tornando-se uma ótima opção para operar em dispositivos com recursos mais sofisticados, como os da IoT [Hahm et al. 2015].

**Raspbian**[5]**.** É um SO gratuito otimizado para o hardware Raspberry Pi, baseado no Debian. Vale ressaltar que o SO vem com mais de 35.000 pacotes e softwares pré-compilados para facilitar a instalação no Raspberry Pi. O Raspian suporta autenticação, autorização e criptografia para multimídia [Bansal and Kumar 2020].

**Contiki**[6]**.** O Contiki é um sistema operacional de código aberto, sob a licença BSD-3-Clause[7], desenvolvido para sistemas em rede com restrições de memória e foco em dispositivos da IoT sem fio de baixo consumo. A comunicação se dá através de protocolos padrão, como IPv6 / 6LoWPAN, 6TiSCH, RPL e CoAP [Hahm et al. 2015]. O Contiki é utilizado em vários projetos comerciais e não comerciais, como na iluminação pública, monitoramento de som para cidades inteligentes, medidores de energia elétrica em rede, dentre outras[8]. O Contiki fornece através do ContikiSec uma camada que provê serviços de segurança: autenticação, sigilo e integridade [Bansal and Kumar 2020].

[1]Disponível em: `https://github.com/android`
[2]Disponível em: `https://github.com/RIOT-OS/RIOT`
[3]Disponível em: `https://www.riot-os.org/`
[4]Disponível em: `https://nuttx.apache.org/`
[5]Disponível em: `https://www.raspbian.org/`
[6]Disponível em: `https://github.com/contiki-os`
[7]Disponível em: `https://opensource.org/licenses/BSD-3-Clause`
[8]Disponível em: `https://github.com/contiki-os/contiki`

**TinyOS**[9]**.** Do mesmo modo que o Contiki, o TinyOS é um SO de código aberto licenciado pelo BSD e projetado para dispositivos sem fio de baixa potência, como os utilizados em redes de sensores, computação ubíqua, redes de área pessoal, edifícios inteligentes e medidores inteligentes [TinyOS 2013]. Além disso, o TinyOS possui suporte a biblioteca TinySec que fornece autenticação de mensagem, integridade e segurança na semântica de confidencialidade [Bansal and Kumar 2020]. Os microcontroladores do primeiro satélite da Estônia (ESTCube-1) utilizaram o TinyOS no módulo de comunicação do satélite [Sünter et al. 2016].

Ademais, sabe-se que os dispositivos e as aplicações da IoT crescem diariamente. Para citar alguns exemplos, destacamos os *smartphones*—compostos por sensores—e as lâmpadas inteligentes ou *smart lights*, que fornecem eficiência energética, conveniência e segurança. Há também aplicações IoT voltadas para o transporte, como os carros inteligentes, ou *smart cars*, que possuem autonomia para se movimentar, e os tratores inteligentes, ou *smart tractors*, que são capazes de realizar tarefas pré-programadas sem a necessidade de um operador. Outras aplicações IoT consistem em casas/edifícios inteligentes, ou *smart homes/buildings*, que possuem sistemas avançados para, por exemplo, controlar a temperatura [De Farias et al. 2019]. Mais dispositivos inteligentes são ilustrados na Figura 7.3.


**Figura 7.3. Dispositivos Inteligentes que compõem a IoT.**

Observa-se que a maior parte dos dispositivos possuem recursos computacionais limitados. Desse modo, a comunicação com tais dispositivos deve obedecer baixo consumo de energia e memória. Dentre as tecnologias que apresentam esses requisitos, destacamos o *Near Field Communication* (NFC), *Wireless Sensor Networks* (WSN), *Zigbee* e *Bluetooth*, que são adotados com frequência pela comunidade [Bansal and Kumar 2020].

### 7.2.4. Arquiteturas de Aplicação de Rede

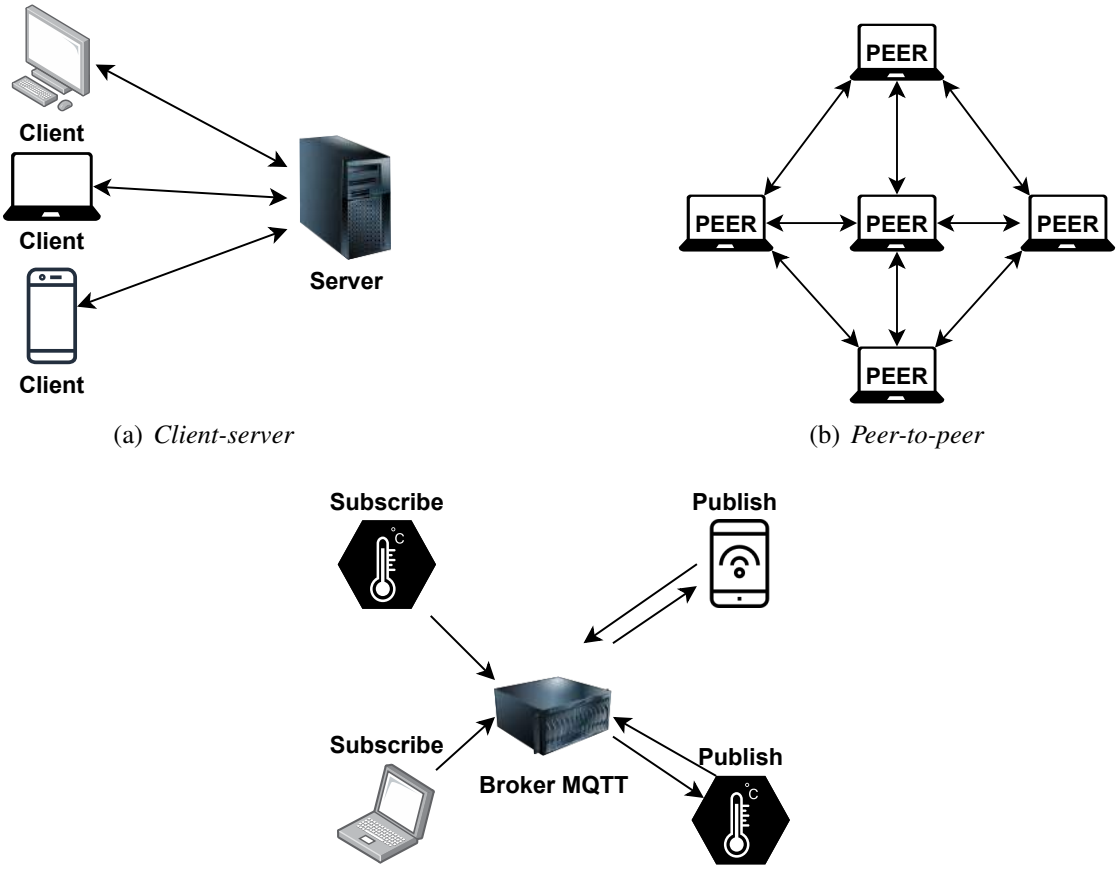
Segundo [Kurose and Ross 2010], uma arquitetura de aplicação de rede determina a forma como a aplicação é organizada nos sistemas finais, além de ser projetada pelo desenvolvedor da aplicação. É diferente de arquitetura de rede, que é algo engessado do ponto de vista do desenvolvedor, fornecendo um conjunto específico de serviços às aplicações.

Essas arquiteturas definem a forma de comunicação entre as partes envolvidas. As arquiteturas de aplicação de rede mais adotadas na camada de aplicação da IoT são

[9]Disponível em: `https://github.com/tinyos`

[Kurose and Ross 2010]: *client/server*, exibido na Figura 7.4(a); e *pear-to-pear*, mostrado na Figura 7.4(b);

Outra arquitetura de aplicação a qual pode ser considerada como tal considerando o conceito acima é a *publish/subscribe*, ilustrada na Figura 7.4(c). Isso porque ela determina a forma como a aplicação está organizada nos sistemas finais: o *broker* sempre ativo e com maiores recursos computacionais, enquanto que o publicador e o assinante permanecem com a conexão intermitente. A seguir, tais arquiteturas são comentadas:




(a) *Client-server*

(b) *Peer-to-peer*

(c) *Publish-subscribe*

**Figura 7.4. As Arquiteturas de Aplicação de Rede mais utilizadas na Camada de Aplicação da IoT.**

- ***Client-server.*** Em português, é denominado como cliente-servidor. Essa arquitetura consiste em uma aplicação distribuída que compartilha tarefas e cargas de trabalho entre os provedores de recursos (*i.e.,* servidores) e os demandantes dos serviços (*i.e.,* clientes). Em outras palavras, os clientes solicitam informações ao servidor, enquanto que o servidor responde ao cliente compartilhando recursos. Essa estrutura costuma adotar o paradigma *request/response* na comunicação, que denota a demanda e a resposta por um serviço [Tanenbaum and Wetherall 2011].
- ***Peer-to-peer.*** Esse paradigma também é conhecido como par-a-par ou ponto-a-ponto (P2P). Nele, cada um dos dispositivos que compõem a rede funcionam tanto como cliente quanto como servidor. Desse modo, é possível compartilhar dados e serviços

sem a necessidade de um servidor central. Um exemplo de aplicação P2P pura consiste no Gnutella, que é uma rede de compartilhamento de arquivos em que as buscas são repassadas de um nó para o outro, dispensando um servidor centralizado. Ainda, é importante destacar a possibilidade de arquitetura híbrida formada a partir das arquitetura cliente-servidor e par-a-par. O *torrent* é um exemplo de aplicação que utiliza arquitetura híbrida. Por exemplo, no *torrent* vários usuários podem realizar a conexão e transmitir arquivos, sem a necessidade de uma fonte intermediária para baixar os arquivos [Tanenbaum and Wetherall 2011].

- ***Publish-subscribe.*** Também conhecido por publicação-assinatura, esse paradigma é composto por três componentes principais: um publicador, um assinante e um servidor centralizado (*broker*). Basicamente, o *broker* é composto por tópicos. Os dispositivos publicadores são responsáveis por alimentar os tópicos do *broker*. Já os assinantes, são dispositivos interessados nas informações contidas nos tópicos, que assinam os mesmos para receber tais informações. Vale ressaltar que um *broker* pode conter múltiplos tópicos. Cada um deles pode receber mensagens de diversos dispositivos publicadores, as quais são entregues para todos os dispositivos que assinam aquele tópico [Quincozes et al. 2019].

Os protocolos da camada de aplicação podem utilizar tais arquiteturas expostas acima. Desse modo, é de extrema importância entende-las para auxiliar na tomada de decisões no momento da implementação de novas aplicações IoT. Por exemplo, o protocolo MQTT é baseado na arquitetura *publish-subscribe* [Nebbione and Calzarossa 2020].

## 7.3. Protocolos da Camada de Aplicação

Nesta seção, os protocolos da camada de aplicação da IoT são apresentados, com ênfase nos protocolos MQTT e CoAP, que são considerados mais proeminentes na literatura. Ambos são considerados leves e recomendados para atuar em aplicações com recursos restritos. Por essa razão, apresentam grande adoção por parte da indústria. A Tabela 7.1 resume os protocolos e as suas principais características [Nebbione and Calzarossa 2020].

### 7.3.1. Protocolo CoAP

É um protocolo inspirado no HTTP e especializado para dispositivos com recursos restritos, conforme especificado na RFC 7252 [Shelby et al. 2014]. O protocolo CoAP utiliza o método de requisição/resposta: um cliente faz uma solicitação para a URI e o servidor responde. Na camada de transporte o CoAP adota o protocolo UDP para transferir mensagens [Tariq et al. 2020], considerado mais leve que o TCP. O protocolo define quatro tipos de mensagens utilizadas para fornecer confiabilidade, as quais são apresentadas a seguir: *Confirmable*, *Non-confirmable*, *Acknowledgment* e *Reset*. A Figura 7.5 ilustra exemplos de mensagens CON e NON.

***Confirmable (CON).*** As mensagens do tipo CON visam prover confidencialidade. Desse modo, elas requerem confirmação quando chegam ao seu destino. Ou seja, se nenhum pacote for perdido durante a transmissão, uma mensagem de confirmação do tipo *Acknowledgement* é retornada. Caso contrário, uma mensagem do tipo *Reset* é repassada ao destinatário [Shelby et al. 2014].

***Non-confirmable (NON).*** As mensagens do tipo NON são consideradas não confiáveis, visto que não requerem confirmação. As mensagens NON são interessantes para

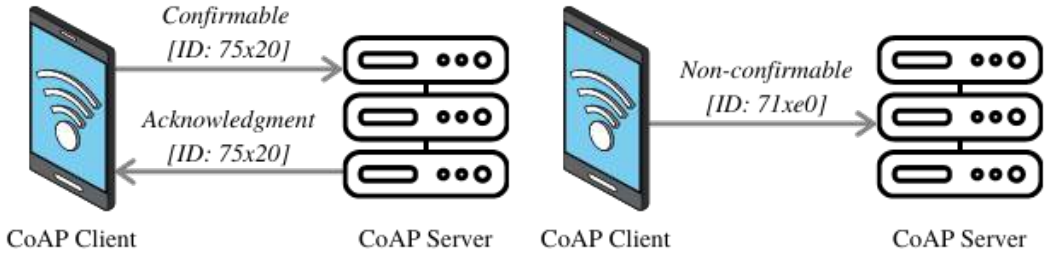
| Protocolo | Característica | Segurança | Transporte | Paradigma |
|---|---|---|---|---|
| **MQTT** | Flexibilidade, simplicidade e confiabilidade na entrega de mensagens. | TLS | TCP | *Publish/Subscribe* |
| **CoAP** | Foco em dispositivos com recursos restritos, alternativa ao HTTP. | DTLS | UDP | *Request/Response*<br>*Publish/Subscribe* |
| **MQTT-SN** | Otimizado para redes de sensores. Possui modo de hibernação. | TLS | UDP | *Publish/Subscribe* |
| **XMPP** | Transferência de mensagens instantâneas (*e.g.,* bate-papos ou videochamadas). | TLS | TCP | *Publish/Subscribe*<br>*Request/Response* |
| **DDS** | Escalabilidade, confiabilidade, comunicação em tempo real - P2P, desempenho. | TLS/DTLS | TCP/UDP | *Publish/Subscribe* |
| **HTTP** | Recursos úteis e difundido. Entretanto, apresenta alto consumo de energia. | SSL | TCP | *Request/Response* |

**Table 7.1. Principais Protocolos da Camada de Aplicação da IoT. Adaptado de [Esfahani et al. 2017] e [Nebbione and Calzarossa 2020].**

aplicações que fazem leituras repetidas regularmente, como as medições em sensores, por exemplo. Uma mensagem do tipo *Reset* é retornada caso o destinatário não seja capaz de processar a mensagem [Shelby et al. 2014].

***Acknowledgment (ACK).*** Uma mensagem do tipo ACK confirma o recebimento de uma mensagem CON específica. A mensagem deve ser identificada por um ID de mensagem (*e.g.,* ACK (ID: 0xVEq51)) [Shelby et al. 2014].

***Reset (RST).*** Uma mensagem do tipo RST indica que uma mensagem do tipo NON foi recebida, mas não pode ser processada corretamente. Normalmente, isso acontece quando um nó receptor é reinicializado e um estado necessário para interpretar a mensagem é esquecido. Da mesma forma que a mensagem do tipo ACK, a RST requer um ID de identificação da mensagem [Shelby et al. 2014].




(a) *Confirmable* - *com confiabilidade.* (b) *Non-confirmable* - sem confiabilidade.

**Figura 7.5. A obtenção de confiabilidade no protocolo CoAP. Adaptado de [Shelby et al. 2014].**

### 7.3.2. Protocolo MQTT

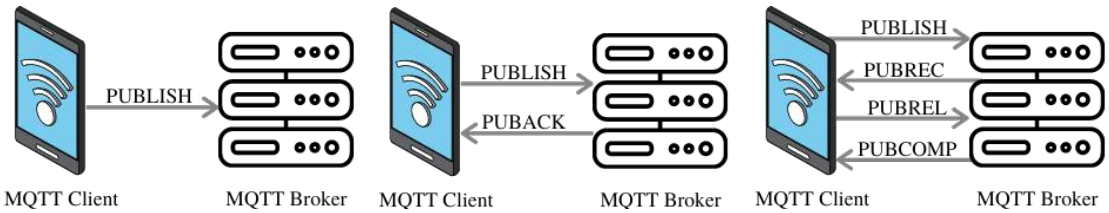
O conceito de MQTT foi proposto por Andy Stanford-Clark e Arlen Nipper, em 1999. Vale ressaltar que em 2013, o MQTT tornou-se o protocolo padrão da organização para

o Avanço dos Padrões de Informação Estruturada (OASIS) [OASIS 2019]. Em síntese, o MQTT utiliza o paradigma de publicação/assinatura para comunicação, e foi projetado para suportar dispositivos com recursos restritos. Basicamente, a arquitetura do MQTT é composta por três tipos de componentes: (1) publicadores, (2) assinantes e um (3) *Broker*. Os dispositivos que desempenham papéis de assinantes são aqueles que se registram em tópicos de seu interesse no *broker*, de modo a receber mensagens transmitidas pelos dispositivos publicadores nos respectivos tópicos. Este paradigma é ilustrado na Figura 7.4(c). Diferente do protocolo CoAP, o MQTT utiliza o protocolo TCP na camada de transporte. Ademais, há três níveis de Qualidade de Serviço, ou em inglês, *Quality of Service* (QoS) do MQTT, que são responsáveis por entregar mensagens e garantir a precisão da comunicação: QoS-0, QoS-1 e QoS-2 [Quincozes et al. 2019].

**QoS-0.** Esse é o nível mais baixo de QoS do MQTT. Nele, não há garantia de que uma mensagem será entregue. Ademais, não é necessário que o destinatário confirme o recebimento da mesma. O QoS 0 é conhecido por "disparar e esquecer" a mensagem, visto que não fornece nenhum aviso ou garantia de entrega [OASIS 2019].

**QoS-1.** No nível de QoS 1, há garantia de que uma mensagem seja entregue pelo menos uma vez ao destinatário. Após disparar a mensagem, a mesma é armazenada pelo remetente até que uma mensagem de retorno do tipo *PUBACK* seja recebida, confirmando o recebimento. Nesse nível, a mensagem pode ser entregue uma ou múltiplas vezes. O pacote *PUBACK* contém um identificador que deve corresponder ao da mensagem *PUBLISH* [OASIS 2019].

**QoS-2.** O nível mais alto de QoS do MQTT é o 2. Nele, uma mensagem é entregue exatamente uma vez. Por isso, o QoS 2 é o mais seguro e, consequentemente, o mais lento. Existem pelo menos dois fluxos de solicitação para prover garantia de entrega: O remetente envia uma mensagem do tipo *PUBLISH* para o destinatário, que deve verificar a mensagem e responder ao remetente com um pacote *PUBREC*. Após obter a resposta, o remetente descarta a primeira mensagem *PUBLISH*, armazena o *PUBREC* recebido do destinatário e responde com um pacote *PUBREL*. A partir do momento que o destinatário obtém o pacote *PUBREL*, ele pode descartar todos os estados anteriores e responder o remetente com uma mensagem do tipo *PUBCOMP*. Da mesma maneira que o destinatário, quando o remetente recebe o *PUBCOMP*, ele pode descartar os estados anteriores. Todas as mensagens possuem um identificador do pacote [OASIS 2019].




(a) *QoS-0.* (b) *QoS-1.* (c) *QoS-2.*

**Figura 7.6. Os níveis de confiabilidade do protocolo MQTT. Essa é uma versão simplificada, visto que as mensagens de estabelecimento e fechamento de conexão estão omitidas. Adaptado de [OASIS 2019].**

A Figura 7.6 ilustra a troca de mensagens em cada nível de confiabilidade do

MQTT. Por fim, vale ressaltar que o MQTT possui duas especificações: o MQTT v5.0 e o MQTT-SN, explanado a seguir.

### 7.3.3. Protocolo MQTT-SN

O MQTT-SN é uma versão adaptada do MQTT, específica para redes de sensores sem fio e com baixa largura de banda. Diferente do MQTT, o MQTT-SN utiliza o *User Datagram Protocol* (UDP) na camada de transporte para transferir mensagens. Uma das funcionalidades do MQTT-SN consiste na possibilidade de permanecer por determinado período de tempo em um modo chamado *sleeping*, onde o *broker* é responsável por armazenar todas as mensagens destinadas ao mesmo [Yassein et al. 2017] [Quincozes et al. 2019].

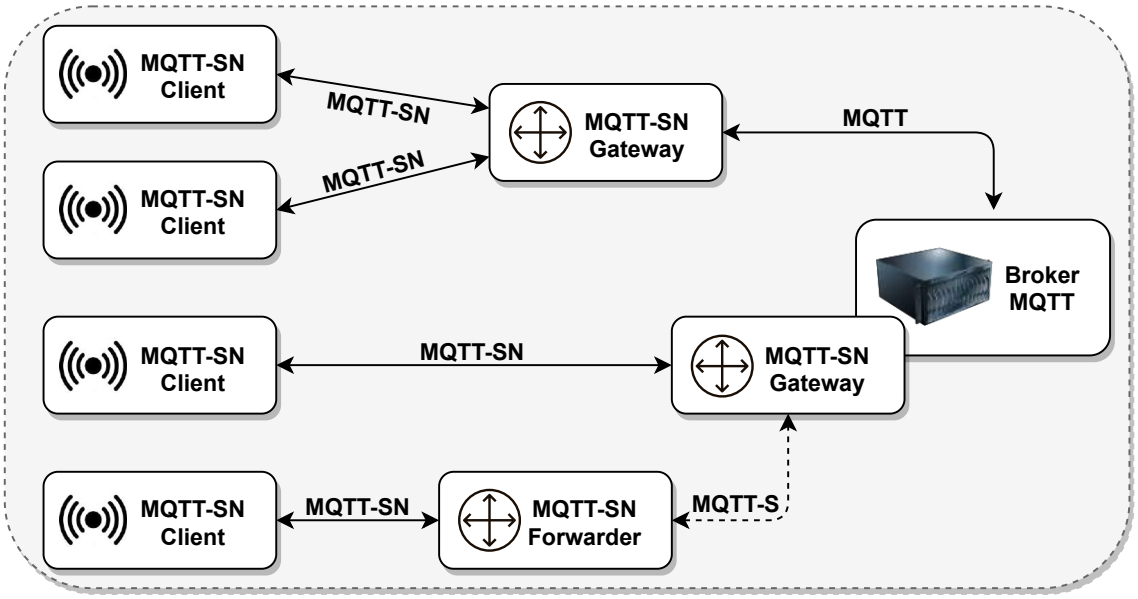



**Figura 7.7. Arquitetura do MQTT-SN. Adaptado de [Stanford-Clark and Truong 2013].**

A arquitetura do MQTT-SN é ilustrada na Figura 7.7. Tal arquitetura é composta por 3 tipos de componentes: MQTT-SN *Client*, MQTT-SN *Gateway* e MQTT-SN *Forwarders*. Normalmente, os *Clients* conectam-se ao *Broker* MQTT por meio dos *Gateways* utilizando o protocolo MQTT-SN. Vários *Clients* podem iniciar a conexão com um *Gateway* específico. Em alguns casos, o *Client* pode acessar o *Broker* através de *Forwarders*. Os *Forwarders* são responsáveis por encapsular os pacotes MQTT-SN e enviar para o *Gateway* [Stanford-Clark and Truong 2013].

### 7.3.4. Extensible Messaging and Presence Protocol (XMPP)

O XMPP é um protocolo aberto, extensível e adaptável, que surgiu em 1999 e teve sua primeira versão publicada em 2000. O XMPP se baseia no protocolo *Extensible Markup Language* (XML) para comunicação e transferência de arquivos em redes distribuídas. É um protocolo projetado para aplicações que exigem comunicação e mensagens instantâneas, como bate-papo em grupo ou vídeo-chamada e áudio. Além disso, possui uma funcionalidade que indica o status atual do cliente (*e.g.* online, offline ou ocupado)

[Çorak et al. 2018]. Nos últimos anos, o XMPP mostrou-se adequado para a IoT, reconquistando a atenção de muitos. O XMPP é executado sobre o TCP, suporta comunicação entre cliente-servidor e cliente-cliente, além disso, pode ser assíncrono utilizando o paradigma de publicação/assinatura ou síncrono através do modelo de solicitação/resposta. Por ser um protocolo projetado para aplicações quase em tempo real, o XMPP suporta mensagens de tamanho reduzido e de baixa latência [Karagiannis et al. 2015]. Empresas de grande porte utilizam o XMPP nas suas aplicações (*e.g. Google Talk*, *Whatsapp*, *Fortnite* e *League of Legends*).

### 7.3.5. Data Distribution Service (DDS)

A especificação do protocolo DDS foi aprovada pelo *Object Management Group* (OMG) em 2004, resultando na publicação da sua primeira versão. O DDS é um protocolo projetado para utilizar o paradigma de publicação/assinatura para comunicações *machine to machine* (M2M), que visa transmitir dados de forma escalável e em tempo real. Ao contrário do protocolo MQTT, o DDS adota a arquitetura *brokerless*, ou seja, sem a existência de nó intermediário. Ademais, ele utiliza *multicast* para fornecer comunicação confiável, eficiente e transmissão de informações para múltiplos destinatários simultaneamente, adaptando-se bem às aplicações IoT e M2M. No DDS, existe um conjunto com 23 políticas de QoS, que proporcionam a implementação de uma variedade de critérios nas comunicações, tais como: segurança, urgência, prioridade, durabilidade, propriedade, orçamento de latência, entre outras [Al-Fuqaha et al. 2015].

### 7.3.6. HTTP

Por fim, vale ressaltar a existência do protocolo HTTP, um dos principais protocolos utilizados em Sistemas de Informação para a transferência de dados utilizando o estilo arquitetural *REpresentational State Transfer* (REST) [Phung et al. 2020]. Embora seja um protocolo maduro, ele não é otimizado para IoT. Por exemplo, o CoAP difere do REST por adotar o protocolo UDP na camada de transporte, tornando-o mais apropriado para aplicações que utilizem dispositivos com recursos restritos na IoT. No entanto, o REST ainda é adotado por diversas plataformas devido a recursos úteis como negociação de tipo de conteúdo, mecanismos de autenticação e *cashing* [Karagiannis et al. 2015].

A primeira versão do protocolo HTTP foi publicada na década de 1990, sob coordenação da *World Wide Web Consortium* (WWWC) e da *Internet Engineering Task Force* (IETF). O protocolo HTTP é comumente utilizado na Internet e é a base para efetivar a comunicação de dados da *World Wide Web* (WWW). O HTTP utiliza o paradigma de requisição-resposta e a comunicação cliente-servidor [Santos et al. 2016]. São definidos oito métodos que indicam a ação que deve ser realizada pelo servidor em uma requisição: *GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS e CONNECT*. Vale ressaltar que ao menos os métodos *GET* e *HEAD* devem ser implementados em um servidor HTTP para operação [Kurose and Ross 2010].

## 7.4. Prática

Nesta seção, a parte prática do minicurso é apresentada. Será dada ênfase na demonstração e configuração de implementações dos protocolos MQTT e CoAP, onde são disponibilizados exemplos de código-fonte com implementações de clientes utilizando bibliotecas

Java, como HiveMQ (para o protocolo MQTT) e CoAP Blaster (para o protocolo CoAP). Tais bibliotecas foram utilizadas para a implementação de um aplicativo *Android*, que é executado em um *smartphone*. Ademais, utilizou-se o *broker* MQTT Mosquitto para uma instalação local, em um dispositivo Raspberry Pi, que recebe mensagens publicadas pelo aplicativo instalado no *smartphone*. Por fim, um *laptop* foi utilizado para o desenvolvimento do aplicativo Android e monitoração de uso de recursos.

### 7.4.1. Implementação de Protótipo de clientes MQTT e CoAP em Android

De modo a demonstrar de maneira prática a implementação de dispositivos clientes CoAP e MQTT, foi implementado um protótipo através de um aplicativo Android que suporta ambos os protocolos. Tal protótipo é ilustrado na Figura 7.8, o qual é disponibilizado publicamente pelos autores deste manuscrito[10]. Para suportar os protocolos estudados, implementou-se um cliente publicador MQTT por meio da biblioteca disponibilizada em java HiveMQ (`com.hivemq-mqtt-client`) [HiveMQ 2012] e um cliente CoAP através da biblioteca CoAP Blaster (`com.google.iot.coap`) [CoapBlaster 2018].

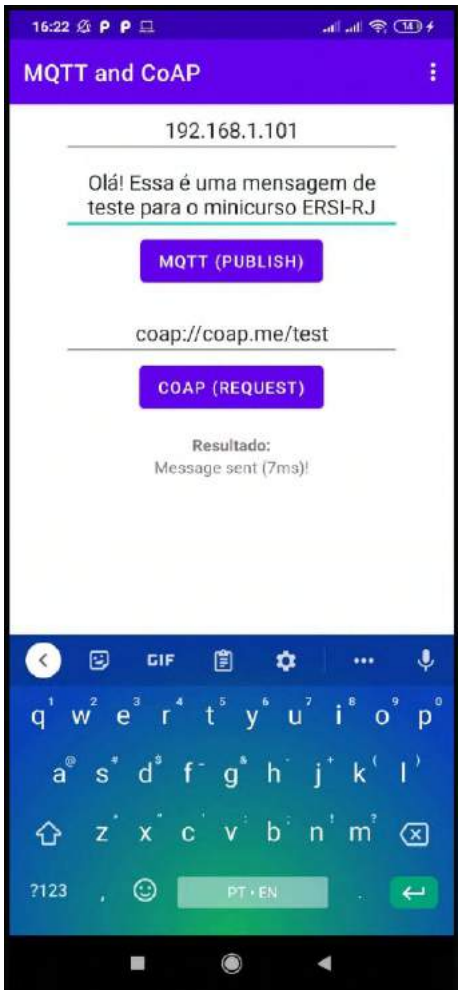
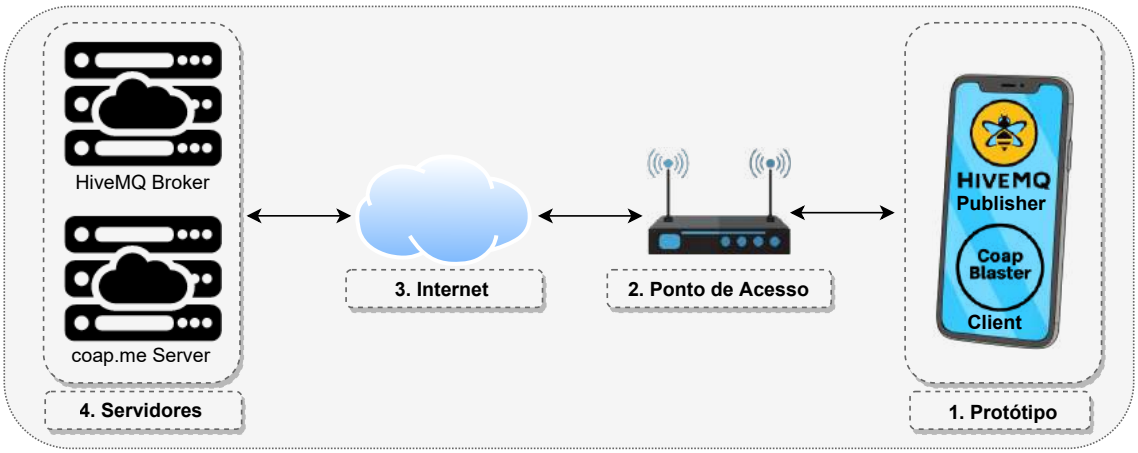



**Figura 7.8. Implementação de protótipo de app Android para testes.**

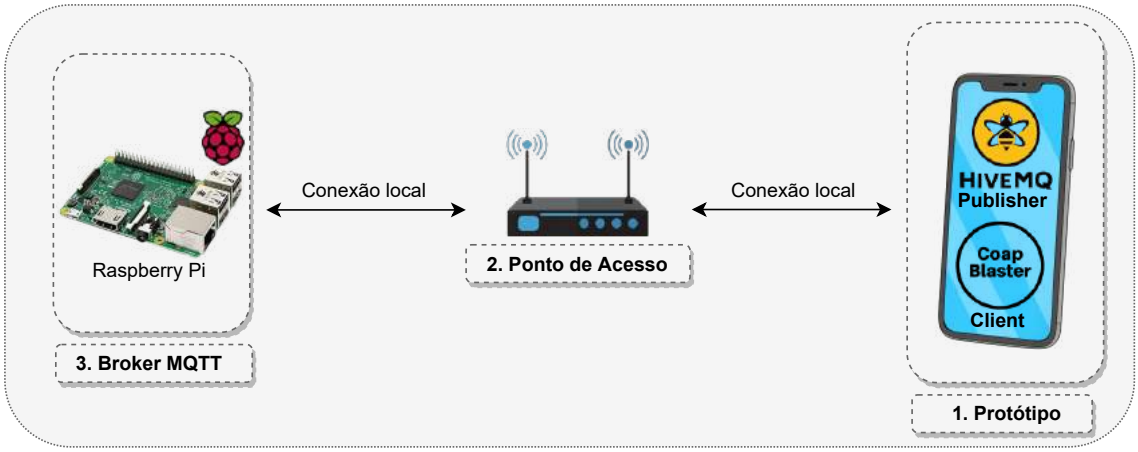
[10]Disponível em: `https://github.com/sequincozes/ERSI-RJ.git`

Tal protótipo inclui uma única tela com campos para o usuário informar o endereço do servidor CoAP ou broker MQTT. Desse modo, através dos botões `MQTT (Publish)` e `CoAP (Request)`, os códigos que implementam a publicação de uma mensagem MQTT (listado no Código-fonte 7.1) e envio de uma requisição CoAP (listado no Código-fonte 7.2), respectivamente, são acionados.

Para fins de validação do protótipo, executaram-se testes, conforme a Figura 7.9(a), através da comunicação do *smartphone* contendo o aplicativo instalado com um *broker* MQTT e de um servidor CoAP publicamente disponíveis. Em síntese, a Figura 7.9(a) ilustra o primeiro cenário, no qual o protótipo (1) se associa ao ponto de acesso sem fio (2), conectando-se à Internet (3) a fim de se comunicar com o *broker* MQTT e o CoAP Server (4) públicos e disponíveis. Adicionalmente, a Figura 7.9(b) ilustra um cenário de validação em rede local. Desse modo, o protótipo (1) desenvolvido comunica através de um ponto de acesso sem fio (2) com *broker* MQTT, o qual é instalado em um dispositivo Raspberry Pi (3). O cenário local é discutido na Seção 7.4.2.




(a) *Cenário 1: Broker MQTT e Servidor CoAP acessados em servidores publicamente acessíveis.*




(b) *Cenário 2: Broker MQTT instalado em um Raspberry Pi.*

**Figura 7.9. Cenários da implementação prática.**

O código-fonte 7.1 ilustra a implementação prática de um cliente MQTT na linguagem de programação Java. As importações dos pacotes necessários são feitas nas

linhas 1 a 6. A classe MqttClient é definida na linha 8 e se estende até a linha 27. Nas linhas 11 a 17, o construtor da classe MqttClient é estabelecido. Também, existe um método responsável por publicar mensagens (linhas 19 a 26).

Perceba que a linha 1 do código-fonte 7.1 importa a biblioteca MqttQos, que é utilizada nas linhas 21 e 22 para definir o nível de confiabilidade da mensagem. Em síntese, três níveis de confiabilidade poderiam ser adotados, conforme visto anteriormente na Figura 7.6: AT_MOST_ONCE que representa o QoS-0, AT_LEAST_ONCE que implementa o QoS-1 e EXACTLY_ONCE, que faz referência ao QoS-2.

**Código-fonte 7.1:** Implementação de um Cliente MQTT em Java.

```
1  import com.hivemq.client.mqtt.datatypes.MqttQos;
2  import com.hivemq.client.mqtt.mqtt3.Mqtt3BlockingClient;
3  import com.hivemq.client.mqtt.mqtt3.Mqtt3Client;
4  import java.io.IOException;
5  import java.util.UUID;
6  import java.util.concurrent.TimeoutException;
7
8  public class MqttClient {
9     Mqtt3BlockingClient client;
10
11    public MqttClient (String uri) {
12       client = Mqtt3Client.builder()
13       .identifier(UUID.randomUUID().toString())
14       .serverHost(uri)
15       .buildBlocking();
16       client.connect();
17    }
18
19    public String publishMessage(String msg) {
20       long beginFullTime = System.currentTimeMillis();
21       client.publishWith().topic("test/topic").qos(MqttQos.
22       AT_LEAST_ONCE) .payload(msg.getBytes()).send();
23       long fullTime = System.currentTimeMillis() - beginFullTime;
24       client.disconnect();
25       return "Message sent (" +fullTime + "ms)!";
26    }
27 }
```

O código-fonte 7.2 refere-se a implementação prática de um cliente CoAP na linguagem de programação Java. Nele, as importações de pacotes são definidas nas linhas 1 a 9. Ressalta-se a existência de importações para exibir informações ao usuário (por exemplo, `Toast`), importações para tratar exceções de erros (por exemplo, `IOException` e `TimeoutException`) e importações específicas do CoAP, como `Client` que refere-se ao cliente a ser instanciado e `Message` que é uma biblioteca que da suporte ao envio e recepção de mensagens.

Ademais, a classe principal, denominada `CoAP Client` tem início na linha 11 e término na linha 38. Tal classe é composta pelo seu construtor (linhas 14 a 23) e pelo método responsável pelo envio de mensagens (linhas 24 a 38).

**Código-fonte 7.2:** Implementação de um Cliente CoAP em Java.

```
1  import android.widget.Toast;
2  import com.google.iot.coap.Client;
3  import com.google.iot.coap.HostLookupException;
4  import com.google.iot.coap.LocalEndpointManager;
5  import com.google.iot.coap.Message;
6  import com.google.iot.coap.RequestBuilder;
7  import com.google.iot.coap.UnsupportedSchemeException;
8  import java.io.IOException;
9  import java.util.concurrent.TimeoutException;
10
11 public class CoAPClient {
12    RequestBuilder requestBuilder;
13
14    public CoAPClient(String uri) {
15       LocalEndpointManager manager = new LocalEndpointManager();
16          try {
17             Client client = new Client(manager, uri);
18             RequestBuilder = client.newRequestBuilder().setConfirmable(true);
19          } catch (UnsupportedSchemeException e) {
20             e.printStackTrace();
21          }
22    }
23
24    public String sendRequest(){
25       Message response = null;
26       try {
27          long beginFullTime = System.currentTimeMillis();
28          response = requestBuilder.send().getResponse();
29          long fullTime = System.currentTimeMillis() - beginFullTime;
30          return "Response received ("+fullTime+"ms):
31          "+response.getPayloadAsString();
32       } catch (InterruptedException | HostLookupException | IOException |
33       TimeoutException e) {
34          e.printStackTrace();
35       }
36       return response.getPayloadAsString();
37    }
38 }
```

A ferramenta *Android Profiler* (ilustrada na Figura 7.10), do Ambiente de Desenvolvimento Integrado, do inglês, *Integrated Development Environment* (IDE) *Android Studio* permite a análise do uso de recursos das aplicações Android. Dessa forma, ao executar cada protocolo (*i.e.,* CoAP e MQTT) é possível analisar o consumo de rede, CPU, memória e energia. Adicionalmente, através do método *System.currentMilisseconds()*, nativo da linguagem Java, o tempo de resposta pode ser medido programaticamente.

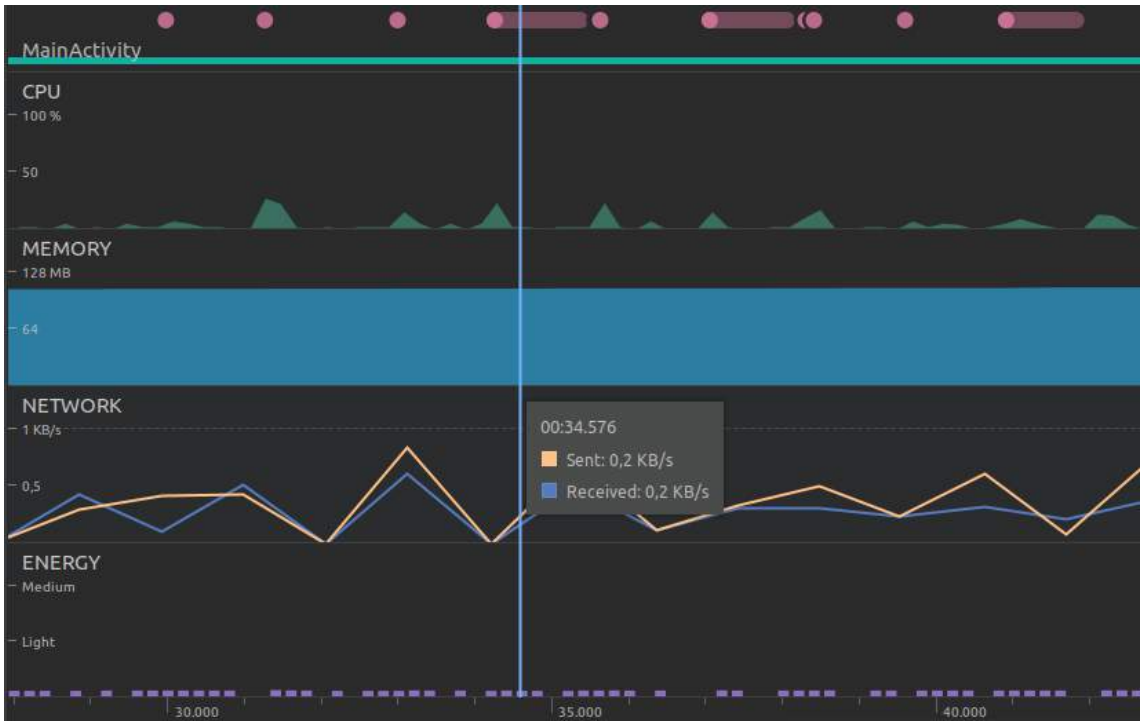



**Figura 7.10. Android Profiler: Ferramenta de análise de uso de recursos do Android Studio.**

### 7.4.2. Instalação e Demonstração de Uso de Broker MQTT em Raspberry Pi

A Figura 7.9(b) ilustra o cenário de rede local configurado para esta demonstração. De modo a instalar o *broker* Mosquitto no sistema operacional Raspbian, cujo é o sistema operacional padrão do Raspberry Pi, foram executados os seguintes comandos:

- Comando para atualização de pacotes e instalação do *broker* Mosquitto e seus clientes (publicador e assinante):

```
pi@raspberry:~ $ sudo apt update
sudo apt install -y mosquitto mosquitto-clients
```

- Comando para inicialização do serviço Mosquitto:

```
pi@raspberry:~ $ sudo systemctl enable mosquitto.service
```

- Comando para checar se o serviço está em execução:

```
pi@raspberry:~ $ mosquitto -v
```

- Comando para consultar o endereço IP do *broker* Mosquitto:

```
pi@raspberry:~ $ hostname -I
```

- Comando para executar um cliente assinante no Mosquitto:

```
pi@raspberry:~ $ mosquitto_sub -d -t testTopic
```

- Comando para executar um cliente publicador no Mosquitto, onde a mensagem `test` é enviada:

```
pi@raspberry:~ $ mosquitto_pub -d -t testTopic -m "test"
```

Note que através da instalação completa do Mosquitto é possível iniciar a execução de um *broker*, publicar mensagens e também assinar a tópicos. No entanto, optou-se pelo uso do protótipo apresentado na Seção 7.4.1 para a publicação de mensagens a partir do *smartphone*, através do protocolo MQTT. Na Figura 7.11, é ilustrado o dispositivo cliente que assina o tópico `test/topic` recebendo uma mensagem do tipo `PUBLISH`. É importante perceber que, para fins de demonstração, o assinante do referido tópico é executado na própria máquina onde roda o *broker*. Já a mensagem é publicada no mesmo tópico pelo dispositivo *smartphone*, usando o protótipo implementado.

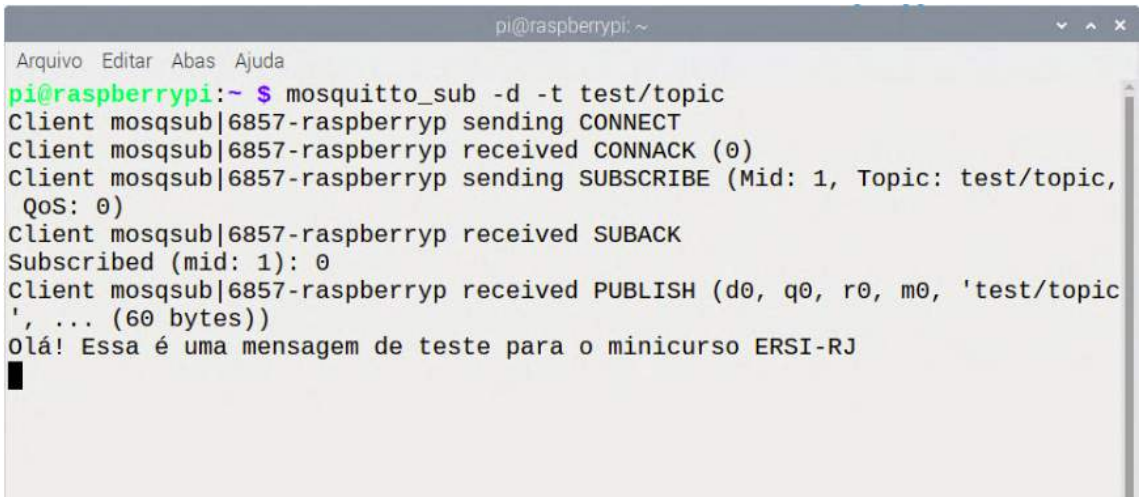



**Figura 7.11. Cliente assinante Mosquitto sendo executado no Raspberry Pi.**

## 7.5. Tendências

Nesta seção, são apresentados assuntos que indicam tendências de estudo na IoT e, em particular, nos protocolos de sua camada de aplicação. Desse modo, são abordados tópicos, como a segurança, a utilização de técnicas de aprendizado de máquina, os sistemas ciber-físicos, os paradigmas computacionais e as interfaces do usuário.

### 7.5.1. Segurança da Informação

A segurança da informação é, cada vez mais, alvo de estudo em um mundo extremamente conectado. Ademais, o advento da IoT vem ampliando significativamente o número de dispositivos ligados à grande rede e impondo desafios no campo da segurança. Nos tempos atuais, intensificou-se a necessidade de resguardar um importante ativo de organizacional que consiste na informação.

Embora existam alternativas convencionais a fim de prover segurança à protocolos da camada de aplicação, como o *Transport Layer Security (TLS)* e o *Datagram TLS (DTLS)*, a adoção dessas opções pode não ser apropriada para dispositivos com recursos limitados, como muitos daqueles existentes no paradigma IoT. Isso tem oportunizado o surgimento de novas propostas envolvendo segurança na literatura especializada, conforme se descreve nos parágrafos seguintes.

Tendo em vista verificar a sobrecarga de mecanismos criptográficos simétricos na camada de aplicação da IoT, os autores de [Quincozes et al. 2021a] propõem estudo que envolve os protocolos CoAP e MQTT. O estudo é dirigido ao uso de cifras simétricas ao invés de cifras assimétricas, uma vez que, segundo os autores, aquelas possuem custo computacional e energético mais adequado para dispositivos com recursos limitado. Desse modo, algoritmos como o *Tiny Encryption Algorithm* (TEA), Data Encryption Standard (DES) e *Advanced Encryption Standard* (AES) foram avaliados através de experimentos práticos disponibilizado publicamente[11]. Os resultados demonstram que o mecanismo TEA é uma boa opção em termos de uso de memória, CPU, consumo energético, tempo de resposta e dados recebidos/enviados. No entanto, ressalta-se que algoritmos como o TEA e o DES estão na lista de cifras não recomendadas por órgãos internacionais, como o *National Institute of Standards and Technology* (NIST) [of Standards and Technology 2014]. Por outro lado, os resultados revelaram que a adoção do algoritmo AES, tanto para o protocolo CoAP quanto para o protocolo MQTT, implicou em maiores sobrecargas em termos de consumo energético e de tempos de resposta, entre outras métricas avaliadas. No entanto, o AES é uma cifra forte e recomendada pelo NIST.

De forma geral, ataques a sistemas de informação ferem a propriedades de segurança como a autenticidade, integridade, disponibilidade, confidencialidade, entre outras [Stallings 2015] [Kurose and Ross 2010] [Tanenbaum and Wetherall 2011]. Por exemplo, ataque de negação de serviço, do inglês, *Denial-of-Service* (DoS) afeta a disponibilidade dos sistemas. Já o ataque da interceptação afeta o sigilo das mensagens. Dessa forma, os trabalhos da literatura propõem mecanismos e arquitetura a fim de mitigar ataques em favor da garantida da propriedades citadas anteriormente.

Em [Rampelotto Junior et al. 2019], os autores propõem um mecanismo denominado *LegimateBroker* com o propósito de mitigar ataques de personificação em *broker* MQTT. A personificação consiste em ataque onde nó malicioso toma o lugar de nó legítimo, geralmente apropriando-se de identidade alheia. A proposta está calcada na autenticação mútua entre Publicadores e *Broker* do MQTT, armazenamento indireto de chaves no *Broker* e renovação periódica de chaves no *Broker* e no Publicador. Experimentos realizados indicam menor sobrecarga do *LegimateBroker* em termos de tempo médio para publicação e consumo energético quando comparado com outras abordagens, por exemplo, o uso de *Transport Layer Security* (TLS).

O trabalho de [Khalil et al. 2020] apresenta um mecanismo que utiliza o protocolo CoAP para a descoberta de recursos (sensores, veículos, atuadores, *smartphones*, etc.) melhorando a autenticação, autorização e controle de acesso a tais recursos. O trabalho considera métricas, como o consumo de CPU, latência, consumo energético. Os experimentos utilizam como recursos um sensor de temperatura e uma tomada elétrica

[11] Disponível em: `https://github.com/sequincozes/ExperimentationAPP2020`.

inteligente. A avaliação de desempenho realizada revelou baixa sobrecarga para todas as métricas retromencionadas quando comparada versão do CoAP que adota o mecanismo de segurança proposto com versão do CoAP que não adota qualquer mecanismo de segurança.

A fim de resistir a ataques como reprodução, *man-in-the-middle*, personificação e modificação, [Esfahani et al. 2017] apresentam um mecanismo baseado em operações ou-exclusivo e resumos criptográficos para autenticação mútua de dispositivos, entre outras propriedades de segurança, em redes IoT. Os autores defendem, através de sua análise de segurança, que o mecanismo provoca baixa sobrecarga em termos de custo computacional e de comunicação e, portanto é apropriado para dispositivos com recursos limitados usados na IoT, como sensores. Também, argumentam que sua proposta se aplica a comunicações *Machine-to-Machine* (M2M) em ambiente de IoT industrial. Para tanto, introduzem os protocolos da camada de aplicação da IoT—MQTT, AMQP, CoAP, XMPP, DDS—como protocolos M2M. A proposta prevê ainda a adoção de *Trusted Platform Module* (TPM) para proteção de chaves criptográficas em dispositivos. Embora o uso de TPM tenda a encarecer os dispositivos, especialmente em redes com larga escala, os autores demonstram preocupação com um importante tópico da segurança da informação: a gerência de chaves. Ela envolve aspectos como a geração, distribuição, armazenamento, renovação e destruição de chaves criptográficas [Menezes et al. 1996].

### 7.5.2. Aprendizado de Máquina

O Aprendizado de Máquina, do inglês, *Machine Learning* (ML) é uma técnica da Inteligência Artificial (IA) que treina máquinas utilizando algoritmos e é capaz de alterar o seu comportamento considerando apenas a experiência própria, em vez de programá-los, necessitando de pouca intervenção humana [Russell and Norvig 2013]. Desse modo, o ML vem sendo uma alternativa promissora para detectar ataques maliciosos e identificar comportamentos anormais através da análise dos mesmos. Recentemente, o ML tem sido amplamente adotado para o aperfeiçoamento de sistemas IoT, especialmente no que diz respeito a segurança da informação, dentre outras áreas [Tahsien et al. 2020].

Um importante direcionamento de pesquisas na área de ML consiste na adoção de suas técnicas para detectar ataques a protocolos da camada de aplicação da Internet das coisas [Tahsien et al. 2020]. Em particular, os protocolos desta camada estão suscetíveis a diversos ataques, como os de negação de serviço (DoS). Esse ataque objetiva consumir todos os recursos disponíveis de um sistema (e.g., memória e processamento). Com a sobrecarga gerada pelo ataque, normalmente a execução dos sistemas é interrompida, tornando-os indisponíveis aos usuários. Na literatura, existem trabalhos que propõem soluções para proteger os protocolos contra esse tipo de ataque. Por exemplo, [Syed et al. 2020] propõem um *framework* baseado em ML para detecção de ataques DoS dirigidos a *broker* MQTT. Os autores testaram a eficiência de sua proposta em três *brokers* MQTT *open-source*: Eclipse Mosquitto, VerneMQ e EMQ. O conjunto de dados (*dataset*) é produzido pelos próprios autores, além de não ser publicizado. Ele é composto por dados relativos a 33 *features* específicas do protocolo MQTT. Os experimentos realizados revelam que a proposta reduz as taxas de falsos positivos.

Outra tendência importante que se percebe consiste na crescente proposta e criação de *datasets* especializados para o estudo de ML junto a protocolos da camada de

aplicação da IoT. Uma dessas propostas consiste no MQTTset [Vaccari et al. 2020]. Nela, os autores propõem um *dataset* com foco no MQTT, composto por diferentes dispositivos da IoT (*i.e.,* sensores de movimento, temperatura e umidade). Esse *dataset* contém 11.915.716 amostras coletadas, incluindo tráfegos malignos e legítimos, sendo disponibilizada publicamente[12]. O *dataset* cobre ataques cibernéticos na rede MQTT, como o *flooding Denial of Service (DoS)*, MQTT *Publish flood*, SlowITe, *malformed data* e *brute force authentication*. O *dataset* proposto conta com 33 *features*, sendo que aproximadamente 90% delas são específicas do protocolo MQTT. Os autores avaliaram o *dataset* considerando alguns algoritmos conhecidos de ML, como o *Random Forest* e *Naïve Bayes* (NV). Os resultados obtidos mostram que os algoritmos tem uma precisão de detecção entre 87% e 91%, exceto o NV, onde os resultados ficam entre 64%.

Outro *dataset* proposto na literatura consiste no MQTT-IoT-IDS2020 Dataset, disponibilizado publicamente[13] [Hindy et al. 2021]. Esse *dataset* é gerado com base em uma arquitetura de rede MQTT simulada. A rede é composta por doze sensores, um corretor, uma câmera simulada e um invasor. O *dataset* cobre quatro tipos de ataques: *aggressive scan, UDP scan, sparta SSH brute-force* e MQTT *brute-force attack*. O *dataset* é composto por 44 *features*, divido entre *features* baseadas em pacotes, unidirecional e bidirecional. Ainda, os autores realizam experimentos usando técnicas de ML. Para facilitar a reprodução, os experimentos são disponibilizados publicamente em um repositório no GitHub[14]. Os resultados demonstram que as *features* baseadas em fluxo são mais adequadas para detectar ataques benignos e baseados em MQTT. Um resultado interessante é que a precisão aumentou de 72% para *features* baseadas em pacotes, enquanto que para o fluxo bidirecional e unidirecional ficou em torno de 99%.

Também, existem propostas onde os *datasets* são construídos com acesso mais restrito e com intuito de validar soluções propostas. Por exemplo, em [Hussain et al. 2021] os autores propõem uma estrutura para detectar ataques maliciosos em ambientes associados à saúde e IoT. A ferramenta IoT-Flock foi utilizada para gerar *dataset*. Essa ferramenta envolve dispositivos IoT e é capaz de gerar tráfegos normais e de ataques. Além disso, ela é capaz de gerar os ataques mais recentes e específicos para os protocolos MQTT e CoAP: MQTT DDoS, MQTT *publish flood, brute force,* SlowITE, além de suportar os ataques MQTT *Authentication Bypass Attack*, MQTT *Packet Crafting Attack* e CoAP *Replay Attack*. O *dataset* contem 10 *features*, sendo que 50% delas são específicas do protocolo MQTT. Ainda, para avaliar o desempenho do *dataset*, os autores utilizaram seis algoritmos de classificação de ML. Dentre os algoritmos avaliados, o *Random Forest* obteve o melhor desempenho na detecção de tráfego normal e malicioso, apresentando 99% de precisão, de *recall*, de acurácia e para F1-Score.

### 7.5.3. Sistemas Ciber-Físicos e Redes Industriais

Em um Sistema Ciber-Físico, do inglês, *Cyber-Physical System* (CPS), os objetos físicos são equipados com dispositivos que possuem recursos de sensoriamento, atuação, computação e comunicação. Tais dispositivos coletam dados da camada de percepção e os

[12]Disponível em: `https://www.kaggle.com/cnrieiit/mqttset`

[13]Disponível em: `https://ieee-dataport.org/open-access/mqtt-internet-things-intrusion-detection-dataset`

[14]Disponível em: `https://github.com/AbertayMachineLearningGroup/MQTT_ML`

transmitem para a camada de aplicação, onde os centros de controle tratam dos processos físicos. A camada de aplicação contém aplicações especializadas para tarefas como como monitoramento, análise de dados e aplicações de usuário final [Quincozes et al. 2021a]. A arquitetura CPS tradicional de três camadas é apresentada na Figura 7.12. Protocolos como o MQTT e CoAP podem ser empregados para a transmissão de informações através das camadas representadas nessa figura.

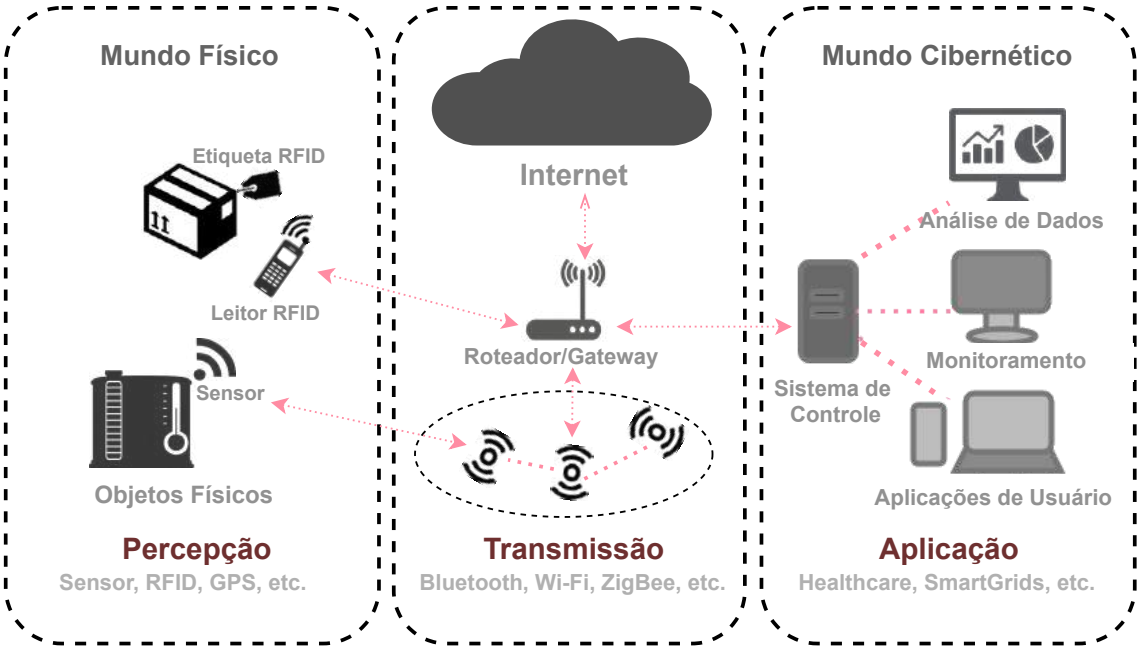



**Figura 7.12. Arquitetura de CPSs (Adaptado de [Quincozes et al. 2021a] e [Santos et al. 2016]).**

A camada de percepção é a principal novidade do CPS em relação aos sistemas de informação tradicionais. Ele contém dispositivos heterogêneos que coletam informações do mundo físico (*e.g.*, sensores e *tags* RFID), identificam certos eventos para responder e atuar de acordo com os mesmos. Esses dispositivos geralmente têm recursos limitados, incluindo a quantidade de memória, capacidade de processamento e fonte de energia [Quincozes et al. 2021a].

Os mundos físico e cibernético podem ser conectados por meio de tecnologias de rede com ou sem infraestrutura, como Wi-Fi, 3G / 4G / 5G, Bluetooth e Zigbee. Portanto, a camada de transmissão também envolve tecnologias e protocolos de comunicação heterogêneos. Esse é um dos principais motivos que leva a adoção de protocolos padronizados como o MQTT [Quincozes et al. 2021a].

Na camada de aplicação, é possível o controle, o monitoramento e o uso personalizado de dados do mundo físico no mundo cibernético. Existem várias aplicações CPS com características diferentes:

- ***Smart Grids*** adicionam monitoramento, controle e comunicação avançados à rede de energia elétrica. Eles permitem o uso eficiente de energia para consumidores, geradores e distribuidores [Quincozes et al. 2021a].

- ***Veículos Inteligentes*** comunicam-se entre si através do que pode ser chamado de CPS Veicular (VCPS). Os VCPSs podem ser utilizados para aumentar a eficiência energética e a capacidade rodoviária, por exemplo, usando uma metodologia baseada em pelotão [Jia et al. 2016], na qual os veículos em um grupo decidem cooperativamente sobre o comportamento de direção [Quincozes et al. 2021a].
- ***Automação Industrial*** é uma tendência que se tornou altamente prevalente e afeta os sistemas industriais modernos. A cooperação entre dispositivos distribuídos e software de controle torna os sistemas de produção mais adaptáveis, versáteis, escaláveis e responsivos. Com a capacidade de reagir em tempo real, os CPSs Industriais (ICPS) são fundamentais no conceito da Indústria 4.0 [Leitão et al. 2016].
- ***Aplicações na Saúde***, onde encontram-se os Health-CPSs (HCPSs) lidam com grandes quantidades de dados complexos de saúde. Portanto, HCPSs devem garantir não apenas a interoperabilidade usando MQTT e CoAP, mas também devem atingir os requisitos de segurança, como privacidade, disponibilidade, autenticidade e integridade [Y. Zhang et al. 2017].

Os impactos dos sistemas ciber-físicos são visíveis na vida cotidiana, principalmente em áreas como elétrica, gás natural, distribuição de petróleo, eletrodomésticos, sistemas de saúde e transporte, dentre outros [Humayed et al. 2017]. Na indústria, os CPS permitem que as informações do espaço físico sejam monitoradas e sincronizadas com o espaço computacional cibernético. As análises avançadas das informações dão margem para as máquinas em rede trabalharem de forma colaborativa e eficiente, gerando uma transformação na indústria de manufatura para a Indústria 4.0 [Lee et al. 2015].

Grande parte dos sistemas ciber-físicos e redes industriais adotam protocolos da camada de aplicação da IoT, como o MQTT e CoAP, para prover interoperabilidade na comunicação. Tais adoções se dão pelas características dos mesmos: são protocolos abertos, simples e leves [Kirchhof et al. 2020]. Ademais, com a versão adaptada do MQTT, o MQTT-SN, existe a possibilidade de determinado dispositivo permanecer inativo por certo período de tempo [Quincozes et al. 2019]. Por isso, torna-se uma ótima opção para a economia de energia em redes industriais.

A Indústria 4.0 refere-se a quarta geração da indústria, que envolve a combinação de diversas tecnologias digitais e visa melhorar a produtividade de tecnologias de fabricação através da coleta e análise de dados em tempo real. Já é comum a projeção de plataformas para produtos e sistemas inteligentes com suporte aos protocolos da camada da aplicação da IoT [Aheleroff et al. 2020], como o MQTT. A Figura 7.13 ilustra a hierarquia entre a IoT, IIoT e Indústria 4.0 [Aazam et al. 2018].

### 7.5.4. Paradigmas computacionais

Os paradigmas computacionais em IoT são utilizados para entregar poder computacional sob demanda para diferentes partes da rede. Desse modo, paradigmas computacionais em IoT de Computação em Nuvem, Computação de Neblina e Computação de Borda, ou, em inglês, respectivamente, *Cloud Computing, Fog Computing* e *Edge Computing* são considerados [Mazon-Olivo and Pan 2021] [Prokhorenko and Babar 2020]. Tais paradigmas estão associados à IoT a fim de prover recursos, como poder de processamento e armazenamento, tendo em vista o grande volume de dados gerados pelos dispositivos de

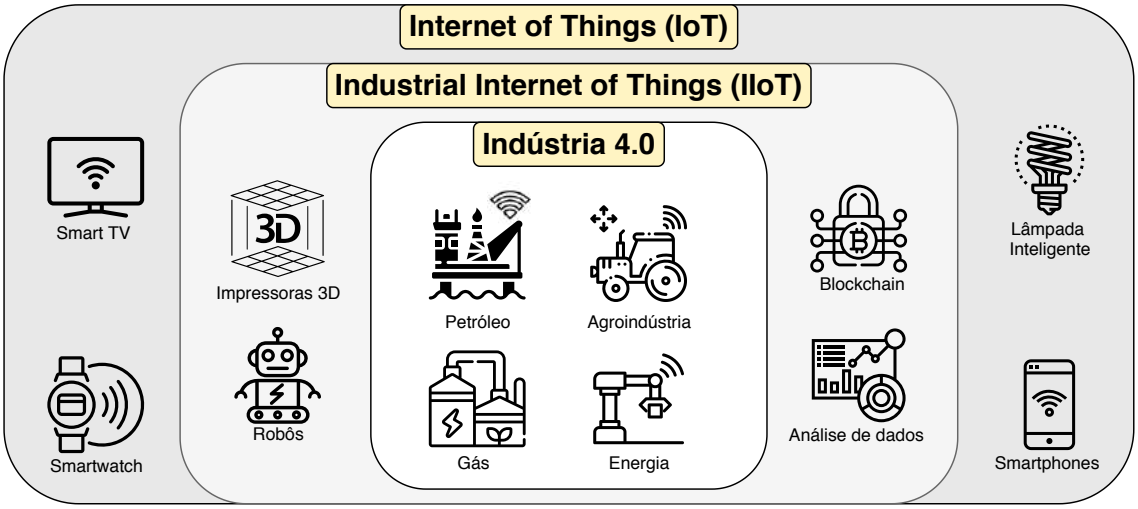



**Figura 7.13. Hierarquia: IoT, IIoT e Indústria 4.0. Adaptado de [Aazam et al. 2018].**

sensoriamento. Embora não sejam propriamente uma novidade, tais paradigmas são de extrema importância para as soluções que envolvam IoT [Mazon-Olivo and Pan 2021].

- **Cloud Computing.** Trata da disponibilização de serviços para empresas. Os principais serviços são processamento e armazenamento de dados em servidores, ou *datacenters*, que são acessíveis pela Internet. Esses serviços tendem a diminuir os custos e aumentar a flexibilidade das empresas [Prokhorenko and Babar 2020].
- **Fog Computing.** Trata-se da junção das soluções em nuvem e borda. O objetivo é promover uma arquitetura descentralizada, de modo em que as aplicações e o gerenciamento fossem distribuídos de forma inteligente entre a fonte de dados e a nuvem [Prokhorenko and Babar 2020]. Também, a *fog computing* vem sendo uma grande aposta para a Indústria 4.0 [Peralta et al. 2017] [Aazam et al. 2018].
- **Edge Computing.** Seu objetivo é semelhante ao da computação em nuvem. A principal diferença é que todo o processamento de dados acontece próximo dos dispositivos IoT, ou seja, na borda da rede onde eles se encontram. Dentre os principais benefícios, destacam-se a redução de banda e latência [Prokhorenko and Babar 2020].

A Figura 7.14 ilustra cada um dos paradigmas. Primeiramente, observa-se que a *Cloud Computing* é composta por milhares de *data centers*. As características deste paradigma consistem na centralização da lógica do negócio, suporte a operação com *big data*, visualização e análise de dados, alta capacidade de computação, dentre outras. A computação em nuvem opera sobre uma arquitetura centralizada. Em contraste com a *cloud computing*, o próximo paradigma, *fog computing*, é composto por milhões de micro *data centers*. Dentre as características deste paradigma, destacam-se a possibilidade de análise de dados, processamento e armazenamento temporal, resposta de controle e controle em tempo real. Este paradigma baseia-se em uma arquitetura distribuída de *fogs*. O último paradigma ilustrado na Figura 7.14, é o *edge computing*. Ele é composto por bilhões de dispositivos da IoT, como sensores e termostatos. Ressalta-se que a *edge computing* abrange sistemas embarcados, micro armazenamento, processamento em tempo real e análises de dados seguindo soluções da inteligência artificial e *machine-learning*.

Da mesma forma que a *fog computing*, esse paradigma opera com uma arquitetura distribuída [Mazon-Olivo and Pan 2021].

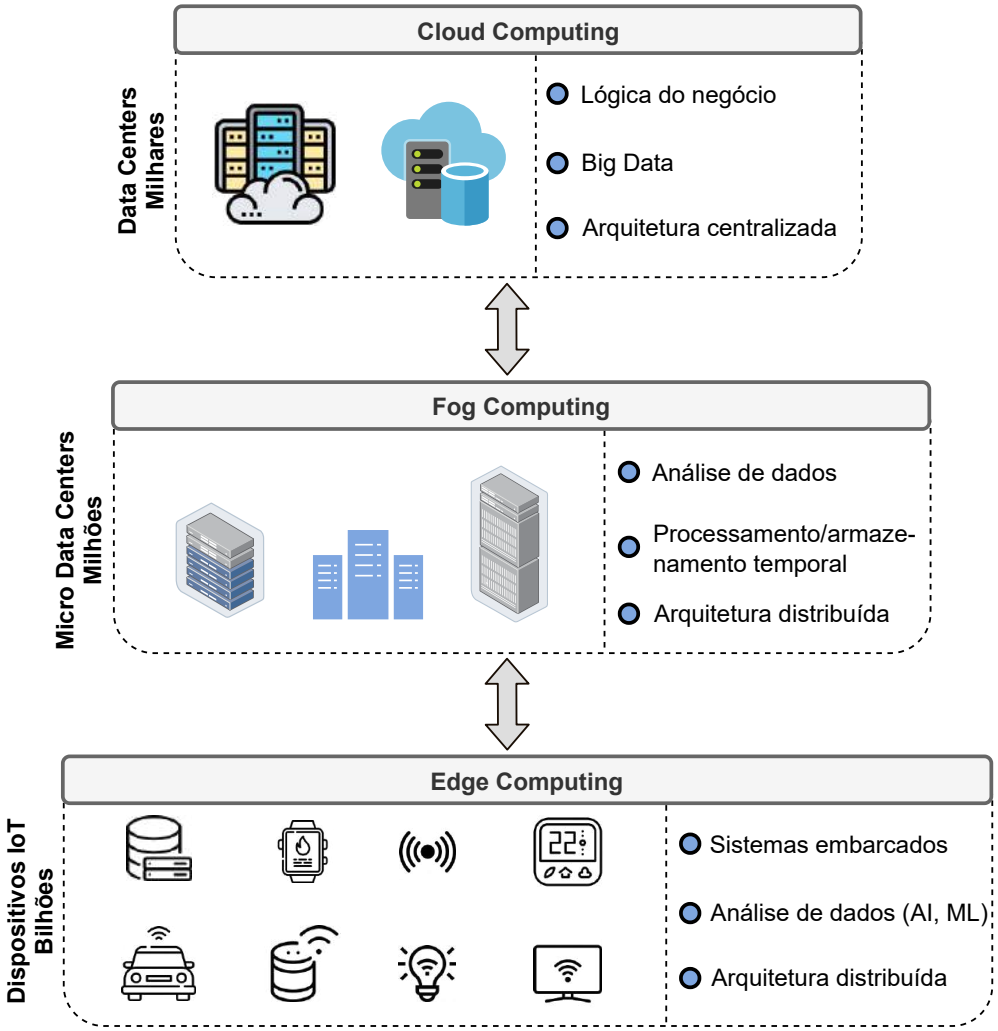



**Figura 7.14. Os paradigmas computacionais *Cloud*, *Fog* e *Edge Computing*. Adaptado de [Mazon-Olivo and Pan 2021].**

### 7.5.5. Interfaces do Usuário para IoT

Tais interfaces permitem gerenciar (*i.e.,* controlar, visualizar e analisar) dados de dispositivos da IoT em tempo real. A conexão dos dispositivos é feita por meio de protocolos da IoT, como o MQTT e o CoAP. Essas interfaces combinam características como escalabilidade, tolerância a falhas e desempenho. Existem inúmeras plataformas gratuitas e de código aberto, como *Bevywise*[15], *ThingsBoard*[16] e *IoTgo*[17].

A ***Bevywise*** é uma plataforma que permite implantar aplicações IoT industriais e comerciais para coletar, analisar e visualizar dados históricos em tempo real. Ela disponibiliza uma API REST para facilitar a construção de aplicações móveis aos clientes. A

[15]Disponível em: `https://www.bevywise.com/iot-dashboard/`
[16]Disponível em: `https://thingsboard.io/`
[17]Disponível em: `https://github.com/itead/IoTgo`

plataforma implementa servidores e *brokers* MQTT e integra funcionalidades de aprendizado de máquina. A *Bevywise* apresenta soluções interessantes para a indústria 4.0, como o monitoramento e gerenciamento de produção, além de soluções futurísticas customizadas para alcançar maior produtividade[18]. Os principais produtos ofertados pela Bevywise são: MQTT *Broker*, *Hosted* MQTT *Broker*, IoT *Platform*, IoT *Simulator*, IoT *Dashboard*, MQTT *Gateway*, *L4Server* - TCP/UDP e IoT *Education Pack*[19]. A Figura 7.15 ilustra um exemplo do painel de monitoramento de produção em tempo real da plataforma.


**Figura 7.15. Painel de Monitoramento de Produção da Plataforma Bevywise. Retirado de [Bevywise 2021].**

A ***ThingsBoard*** é uma plataforma IoT de código aberto para coleta de dados, processamento, visualização e gerenciamento de dispositivos. Os dispositivos adotam os protocolos da camada de aplicação da IoT, como o MQTT, CoAP e HTTP para prover conexão. A plataforma oferece suporte a implantações na nuvem e local, além de permitir a construção de *clusters* para obter o máximo de escalabilidade, tolerância a falhas e desempenho com a implementação da arquitetura de microsserviços. Outra característica dessa plataforma consiste na possibilidade de originar painéis para visualizar dados e controlar remotamente dispositivos da IoT em tempo real. Para isso, a plataforma disponibiliza mais de 30 *widgets* personalizáveis. A ferramenta fornece suporte para aplicações no contexto de energia inteligente (*i.e., smart energy*), agricultura inteligente (*i.e., smart farming*), medições inteligentes (*i.e., smart metering*) e rastreamento de frota (*i.e., fleet tracking*)[20]. A Figura 7.16 ilustra o painel de monitoramento para agricultura inteligente.

Assim como *ThingsBoard*, a ***IoTgo*** é uma plataforma de código aberto que permite que qualquer pessoa implemente o seu próprio serviço de nuvem. Segundo os autores, a plataforma foi projetada para ser aberta, gratuita, simples e de fácil utilização.

[18]Disponível em: `https://www.bevywise.com/industry-4-0/solutions/`
[19]Disponível em: `https://www.bevywise.com/`
[20]Disponível em: `https://github.com/thingsboard/thingsboard`


**Figura 7.16. Painél de Controle Agrícola Inteligente da Plataforma ThingsBoard. Retirado de [ThingsBoard 2021].**

A documentação oficial[21] apresenta as informações necessárias para instalação, configuração e execução da IoTgo, além de uma biblioteca com a relação de dispositivos compatíveis. A plataforma fornece suporte às interfaces *mobile*, *web* e *desktop*. Além disso, ela disponibiliza uma API que, quando integrada às interfaces, permite que os clientes controlem os dispositivos. A plataforma fornece suporte aos protocolos HTTP e WebSocket. No entanto, os criadores recomendam a adoção do WebSocket devido às características: permite o envio da atualização do status do dispositivo tanto para o dispositivo real quanto para os clientes do proprietário do dispositivo.

## 7.6. Conclusão

Neste minicurso, os fundamentos da IoT foram apresentados, desde a fundamentação histórica, arquitetura, sistemas operacionais, dispositivos, arquiteturas de aplicação de rede e, com maior ênfase, a sua camada de aplicação. Protocolos dessa camada como XMPP, DDS e HTTP REST, CoAP e MQTT foram apresentados, sendo os dois últimos alvo de maior destaque. Realizou-se demonstração prática que consistiu na implementação de um protótipo de aplicativo em Android que instância clientes MQTT e CoAP através das bibliotecas HiveMQ [HiveMQ 2012] e CoapBlaster [CoapBlaster 2018]. O protótipo de aplicativo está disponibilizado publicamente pelos autores deste manuscrito[22]. A implementação contou com dois cenários: (*i*) Broker MQTT e Servidor CoAP acessados em servidores publicamente disponíveis na Internet; e *(ii)* Broker MQTT instalado na plataforma Raspberry Pi acessado através do protótipo em ambiente de rede local. No segundo cenário, demonstra-se os comandos para instalação e uso do Broker MQTT.

Adicionalmente, assuntos relevantes que indicam tendências de estudo na IoT e

[21]Disponível em: `https://github.com/itead/IoTgo`
[22]Disponível em: `https://github.com/sequincozes/ERSI-RJ.git`

nos protocolos da camada de aplicação foram apresentados. O primeiro deles consiste na segurança da informação, onde foi indicada a clara necessidade de mecanismos customizados para rodar em dispositivos da IoT. Um segundo assunto trata-se do aprendizado de máquina, que vem sendo uma alternativa promissora para detectar ataques maliciosos nos protocolos da camada de aplicação da IoT. Particularmente, percebe-se forte tendência na construção de *datasets* especializados para o estudo de protocolos da camada de aplicação. O terceiro refere-se aos sistemas ciber-físicos e redes industriais, que estão cada vez mais presentes em aplicações como *smart grids*, veículos inteligentes, automação industrial e saúde. Outro assunto tratado consiste nos paradigmas computacionais *cloud, fog* e *edge computing*, utilizados para entregar poder computacional sob demanda para diferentes partes da rede. Nota-se, que é uma tendência crescente a integração de IoT com tais paradigmas computacionais a fim de melhorar, por exemplo, o tempo de resposta das aplicações, capacidade de armazenamento e processamento. Por fim, as interfaces do usuário para IoT foram apresentadas. Elas permitem controlar, visualizar e analisar dados de dispositivos e têm papel importante uma vez que apresentam os dados oriundos dos dispositivos IoT ao usuário, suportando o uso de protocolos da camada de aplicação.

Espera-se que este documento possa auxiliar no processo de tomada de decisão por parte de desenvolvedores e analistas na implantação de novas aplicações IoT. Para tanto, buscou-se trazer a luz os fundamentos de IoT, bem como os protocolos mais recomendados que operam em nível de camada de aplicação e suas características. Através da implementação prática, espera-se motivar novos usuários e desenvolvedores que estão entrando nesta área. Por fim, acredita-se que a discussão envolvendo assuntos e tendências possa nortear os próximos passos daqueles que buscam ingressar, ou até mesmo os já iniciados, na temática tratada neste minicurso.

## Referências


Aazam, M., Zeadally, S., and Harras, K. A. (2018). Deploying Fog Computing in Industrial Internet of Things and Industry 4.0. *IEEE Transactions on Industrial Informatics*, 14(10):4674–4682.

Aheleroff, S., Xu, X., Lu, Y., Aristizabal, M., Velásquez, J. P., Joa, B., and Valencia, Y. (2020). Iot-enabled smart appliances under industry 4.0: A case study. *Advanced engineering informatics*, 43:101043.

Al-Fuqaha, A., Guizani, M., Mohammadi, M., Aledhari, M., and Ayyash, M. (2015). Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE communications surveys & tutorials*, 17(4):2347–2376.

Alliance, L. (2017). LoRaWAN® Specification v1.1. Disponível em: https://lora-alliance.org/resource_hub/lorawan-specification-v1-1/. Acessado em: Agosto/2021.

Alliance, O. H. (2007). Industry Leaders Announce Open Platform for Mobile Devices. Disponível em: http://www.openhandsetalliance.com/press_110507.html. Acessado em: Agosto/2021.

Alliance, T. Z. (2015). ZigBee Pro Specification - Connectivity Standards Alliance. Disponível em: https://zigbeealliance.org/wp-content/uploads/2019/11/docs-05-3474-21-0csg-zigbee-specification.pdf. Acessado em: Agosto/2021.

Ashton, K. et al. (2009). That 'Internet of Things' Thing. *RFID Journal*, 22(7):97–114.

Bansal, S. and Kumar, D. (2020). Ecosystem: A Survey on Devices, Gateways, Operating Systems, Middleware and Communication. *International Journal of Wireless Information Networks*, 27:340–364.

Bevywise (2021). Bevywise. Disponível em: https://www.bevywise.com/industry-4-0/manufacturing-execution-system/. Acessado em: Agosto/2021.

Borgiani, V., Moratori, P., Kazienko, J. F., Tubino, E. R., and Quincozes, S. E. (2021). Toward a Distributed Approach for Detection and Mitigation of Denial-of-Service Attacks Within Industrial Internet of Things. *IEEE Internet of Things Journal*, 8(6):4569–4578.

CoapBlaster (2018). CoapBlaster. Disponível em: https://github.com/google/coapblaster. Acessado em: Agosto/2021.

Çorak, B. H., Okay, F. Y., Güzel, M., Murt, Ş., and Ozdemir, S. (2018). Comparative Analysis of IoT Communication Protocols. In *2018 International symposium on networks, computers and communications (ISNCC)*, pages 1–6. IEEE.

Cosmi, A. B. and Mota, V. F. (2019). Uma Análise dos Protocolos de Comunicação para Internet das Coisas. In *III Workshop de Computação Urbana*, pages 153–166.

Cox, M. and Ellsworth, D. (1997). Application-Controlled Demand Paging for Out-of-Core Visualization. In *Proceedings. Visualization'97 (Cat. No. 97CB36155)*, pages 235–244. IEEE.

De Farias, C., Rodrigues Caldas de Aquino, G., Costa, G., Kopp, L. F., and Campos, B. (2019). Fusão de dados para Ambientes Inteligentes. In *Livro de Minicursos da VI Escola Regional de Sistemas de Informação do Rio de Janeiro (ERSI-RJ 2019)*, chapter 5, pages 133–157.

Esfahani, A., Mantas, G., Matischek, R., Saghezchi, F. B., Rodriguez, J., Bicaku, A., Maksuti, S., Tauber, M. G., Schmittner, C., and Bastos, J. (2017). A Lightweight Authentication Mechanism for M2M Communications in Industrial IoT Environment. *IEEE Internet of Things Journal*, 6(1):288–296.

Floerkemeier, C., Langheinrich, M., Fleisch, E., Mattern, F., and Sarma, S. E. (2008). *The Internet of Things: First International Conference, IOT 2008, Zurich, Switzerland, March 26-28, 2008, Proceedings*, volume 4952. springer.

Hahm, O., Baccelli, E., Petersen, H., and Tsiftes, N. (2015). Operating Systems for Low-End Devices in the Internet of Things: a Survey. *IEEE Internet of Things Journal*, 3(5):720–734.

Hindy, H., Bayne, E., Bures, M., Atkinson, R., Tachtatzis, C., and Bellekens, X. (2021). Machine Learning Based IoT Intrusion Detection System: An MQTT Case Study (MQTT-IoT-IDS2020 Dataset). In *Selected Papers from the 12th International Networking Conference*, pages 73–84. Springer, Springer International Publishing.

HiveMQ (2012). HiveMQ. Disponível em: https://www.hivemq.com/. Acessado em: Agosto/2021.

Humayed, A., Lin, J., Li, F., and Luo, B. (2017). Cyber-Physical Systems Security – A Survey. *IEEE Internet of Things Journal*, 4(6):1802–1831.

Hussain, F., Abbas, S. G., Shah, G. A., Pires, I. M., Fayyaz, U. U., Shahzad, F., Garcia, N. M., and Zdravevski, E. (2021). A Framework for Malicious Traffic Detection in IoT Healthcare Environment. *Sensors*, 21(9):3025.

Jia, D., Lu, K., Wang, J., Zhang, X., and Shen, X. (2016). A survey on platoon-based vehicular cyber-physical systems. *"IEEE Commun. Surveys Tuts."*, 18(1):263–284.

Karagiannis, V., Chatzimisios, P., Vazquez-Gallego, F., and Alonso-Zarate, J. (2015). A Survey on Application Layer Protocols for the Internet of Things. *Transaction on IoT and Cloud computing*, 3(1):11–17.

Khalil, K., Elgazzar, K., Abdelgawad, A., and Bayoumi, M. (2020). A security approach for CoAP-based internet of things resource discovery. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6.

Kirchhof, J. C., Michael, J., Rumpe, B., Varga, S., and Wortmann, A. (2020). Model-driven Digital Twin Construction: Synthesizing the Integration of Cyber-Physical Systems with Their Information Systems. In *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pages 90–101.

Kurose, J. F. and Ross, K. W. (2010). *Redes de Computadores e a Internet: Uma Abordagem Top-Down.* São Paulo: Addison Wesley, 5.ed.

Lee, J., Bagheri, B., and Kao, H.-A. (2015). A Cyber-Physical Systems architecture for Industry 4.0-based manufacturing systems. *Manufacturing letters*, 3:18–23.

Leitão, P., Colombo, A. W., and Karnouskos, S. (2016). Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges. *Computers in Industry*, 81:11–25.

Mazon-Olivo, B. and Pan, A. (2021). Internet of Things: State-of-the-art, Computing Paradigms and Reference Architectures. *IEEE Latin America Transactions*, 100(1e).

Menezes, A. J., Oorschot, P. C. V., and Vanstone, S. A. (1996). *Handbook of Applied Cryptography*. Boca Raton, FL: CRC Press.

Nebbione, G. and Calzarossa, M. C. (2020). Security of IoT Application Layer Protocols: Challenges and Findings. *Future Internet*, 12(3).

OASIS (2019). MQTT Version 5.0 OASIS Standard. Disponível em: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. Acessado em: Agosto/2021.

OASIS (2020). MQTT for Sensor Networks (MQTT-SN) Version 1.3. Disponível em: https://www.oasis-open.org/committees/download.php/66972/mqtt-sn-v1.3-wd02.docx. Acessado em: Agosto/2021.

of Standards, N. I. and Technology (2014). FIPS 140-2 Security Policy. Disponível em: https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/security-policies/140sp2092.pdf. Acessado em: Agosto/2021.

Pardo-Castellote, G. (2003). OMG Data-Distribution Service: Architectural Overview. In *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.*, pages 200–206. IEEE.

Peralta, G., Iglesias-Urkia, M., Barcelo, M., Gomez, R., Moran, A., and Bilbao, J. (2017). Fog Computing Based Efficient IoT Scheme for the Industry 4.0. In *2017 IEEE international workshop of electronics, control, measurement, signals and their application to mechatronics (ECMSM)*, pages 1–6. IEEE.

Phung, C. V., Dizdarevic, J., and Jukan, A. (2020). An Experimental Study of Network Coded REST HTTP in Dynamic IoT Systems. In *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE.

Postel, J. (1980). RFC 768: User Datagram Protocol. Disponível em: https://rfc-editor.org/rfc/rfc768.txt. Acessado em: Agosto/2021.

Postel, J. (1981). RFC 793: Transmission Control Protocol. Disponível em: https://rfc-editor.org/rfc/rfc793.txt. Acessado em: Agosto/2021.

Prokhorenko, V. and Babar, M. A. (2020). Architectural Resilience in Cloud, Fog and Edge Systems: A Survey. *IEEE Access*, 8:28078–28095.

Quincozes, S., Emilio, T., and Kazienko, J. F. (2019). MQTT Protocol: Fundamentals, Tools and Future Directions. *IEEE Latin America Transactions*, 17(09):1439–1448.

Quincozes, S. E. and Kazienko, J. F. (2020). Machine Learning Methods Assessment for Denial of Service Detection in Wireless Sensor Networks. In *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*, pages 1–6. IEEE.

Quincozes, S. E., Mossé, D., Passos, D., Albuquerque, C., Ochi, L. S., and dos Santos, V. F. (2021a). On the Performance of GRASP-Based Feature Selection for CPS Intrusion Detection. *IEEE Transactions on Network and Service Management*.

Quincozes, V. E., Quincozes, S. E., and Kazienko, J. F. (2021b). Avaliando a Sobrecarga de Mecanismos Criptográficos Simétricos na Internet das Coisas: Uma Comparação Quantitativa entre os Protocolos MQTT e CoAP. In *XX Workshop em Desempenho de Sistemas Computacionais e de Comunicação*, pages 13–24. SBC.

Rampelotto Junior, C., Quincozes, S. E., and Kazienko, J. F. (2019). LegitimateBroker: Mitigando Ataques de Personificação em Broker MQTT na Internet das Coisas. In *XIX Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 141–154. SBC.

Russell, S. J. and Norvig, P. (2013). *Inteligência artificial*. GEN LTC, 3.ed.

Saint-Andre, P. et al. (2004). Extensible Messaging and Presence Protocol (XMPP): Core. Disponível em: https://datatracker.ietf.org/doc/html/draft-ietf-xmpp-3920bis. Acessado em: Agosto/2021.

Santos, B. P., Silva, L. A., Celes, C. S., Borges Neto, J. B., Peres, B. S., Vieira, M. A. M., Vieira, L. F. M., Goussevskaia, O. N., and Loureiro, A. A. (2016). Internet das Coisas: da Teoria à Prática. In *XXXIV Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, chapter 1, pages 1–50.

Shelby, Z., Klaus, H., and Bormann, C. (2014). The Constrained Application Protocol (CoAP). RFC 7252, Universitaet Bremen TZI. Acessado em: Agosto/2021.

Stallings, W. (2015). *Criptografia e Segurança de Redes: Princípios e Práticas*. São Paulo: Pearson, 6.ed.

Stanford-Clark, A. and Truong, H. L. (2013). MQTT for sensor networks (MQTT-SN) protocol specification. *International business machines (IBM) Corporation version*, 1(2).

Sünter, I., Slavinskis, A., Kvell, U., Vahter, A., Kuuste, H., Noorma, M., Kutt, J., Vendt, R., Tarbe, K., Pajusalu, M., et al. (2016). Firmware Updating Systems for Nanosatellites. *IEEE Aerospace and Electronic Systems Magazine*, 31(5):36–44.

Syed, N. F., Baig, Z., Ibrahim, A., and Valli, C. (2020). Denial of service attack detection through machine learning for the IoT. *Journal of Information and Telecommunication*, 4(4):482–503.

Tahsien, S. M., Karimipour, H., and Spachos, P. (2020). Machine Learning Based Solutions for Security of Internet of Things (IoT): A Survey. *Journal of Network and Computer Applications*, 161:18.

Tanenbaum, A. S. and Wetherall, D. (2011). *Computer networks, 5th Edition*. Prentice Hall Professional Technical Reference.

Tariq, M. A., Khan, M., Raza Khan, M. T., and Kim, D. (2020). Enhancements and Challenges in CoAP—A Survey. *Sensors*, 20(21):6391.

ThingsBoard (2021). ThingsBoard. Disponível em: https://thingsboard.io/smart-farming/. Acessado em: Agosto/2021.

TinyOS (2013). TinyOS. Disponível em: http://www.tinyos.net/. Acessado em: Agosto/2021.

Vaccari, I., Chiola, G., Aiello, M., Mongelli, M., and Cambiaso, E. (2020). MQTTset, a New Dataset for Machine Learning Techniques on MQTT. *Sensors*, 20(22):6578.

Weiser, M. (1991). The Computer for the 21st Century. Disponível em: http://wiki.daimi.au.dk/pca/_files/weiser-orig.pdf. Acessado em: Agosto/2021.

Y. Zhang et al. (2017). Health-CPS: Healthcare cyber-physical system assisted by cloud and big data. *IEEE Systems Journal*, 11(1):88–95.

Yassein, M. B., Shatnawi, M. Q., Aljwarneh, S., and Al-Hatmi, R. (2017). Internet of Things: Survey and open issues of MQTT Protocol. In *2017 International Conference on Engineering & MIS (ICEMIS)*, pages 1–6. IEEE.

**Vagner Ereno Quincozes** possui o título de Engenheiro de Software pela Universidade Federal do Pampa (UNIPAMPA) e atualmente é mestrando em Engenharia de Software (UNIPAMPA). Tem interesse nas linhas de pesquisas relacionadas a Segurança da Informação, Redes de Computadores, Provedores de Serviços de Internet, Engenharia de Software, Aprendizado de Máquina e Internet das Coisas. Atualmente participa do projeto de Inovação em Provedores de Serviço de Internet Regionais, no Laboratório de Estudos Avançados (LEA) da UNIPAMPA. Também, participa do projeto de pesquisa e extensão MARFIM - Métodos de Aprendizado de Máquina aplicados à Segurança em Redes Formadas por Dispositivos com Recursos Limitados, na Universidade Federal de Santa Maria (UFSM).
**CV Lattes:** `http://lattes.cnpq.br/5834103446295383`

**Silvio Ereno Quincozes** é professor no IFRS (Canoas). Engenheiro de Software (UNIPAMPA, Brasil), Mestre em Ciência da Computação (UFSM, Brasil) e doutorando em Ciência da Computação (UFF, Brasil), com período sanduíche na University of Pittsburgh (Pitt, EUA). Seus principais interesses de linha de pesquisa incluem Segurança da Informação, Redes de Computadores, Internet das Coisas, Mineração de Dados e Engenharia de Software. Também têm experiências profissionais como Consultor de Sistemas, Desenvolvedor Android, Desenvolvedor PHP e Analista de Dados. Atualmente é bolsista pelo Programa de Excelência Acadêmica da CAPES (Proex). É membro do projeto "Teleproteção em IEC-61850" e do Laboratório Midiacom. Revisor dos periódico IEEE Access e Computers Security, e dos simpósios Simpósio Brasileiro de Telecomunicações e Processamento de Sinais (SBrT) e Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC).
**CV Lattes:** `http://lattes.cnpq.br/9401130360785458`

**Juliano Fontoura Kazienko** possui graduação nos cursos de Administração e de Ciência da Computação pela Universidade Regional Integrada do Alto Uruguai e Missões (URI) e Universidade do Vale do Itajaí (UNIVALI), respectivamente. Concluiu seu mestrado em Ciência da Computação pela Universidade Federal de Santa Catarina (UFSC). Concluiu o doutorado em Computação pela Universidade Federal Fluminense (UFF), com período sanduíche na Universidade da Califórnia em Irvine (UCI), Estados Unidos. Em 2013, recebeu premiação da Secretaria de Assuntos Estratégicos da Presidência da República (SAE/PR) em razão de sua tese de doutorado. Exerceu o cargo de professor adjunto na Universidade Federal do Pampa (UNIPAMPA) no período de 2013 até

2016. Atualmente, é professor adjunto da Universidade Federal de Santa Maria (UFSM). Tem experiência em ciência da computação, especialmente na área de segurança em computação. Seus temas de interesse são: Segurança da Informação, Redes Sem fio e Móveis, Redes Centradas em Informação, Inteligência Artificial aplicada a Redes e Internet das Coisas.
**CV Lattes:** `http://lattes.cnpq.br/7847000086448712`