UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

GABRIEL PRANDO

ARQUITETURA IOT DE ALTA DISPONIBILIDADE E BAIXA LATÊNCIA UTILIZANDO MQTT-SN BASEADA EM COMPUTAÇÃO DE BORDA E NUVEM APLICADA AO CONTEXTO DE CIDADES INTELIGENTES E BIG DATA

PATO BRANCO

2022

LISTA DE CÓDIGOS-FONTE

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A área de IoT (internet das coisas) vem em uma grande crescente na última década, muito devido à expansão da internet e de aplicações distribuídas. Cada vez mais dispositivos inteligentes vêm sendo lançados no mercado, em sua grande maioria conectados a internet e com algum meio de telemetria ou comunicação. Segundo VAILSHERY (2016), estima-se que até meados do ano de 2023 o número de dispositivos conectados atinja a casa dos 50 bilhões e nos próximos anos ultrapasse os 75 bilhões.

A comunicação sem fio é o meio mais utilizado pelos dispositivos IoT, sendo que entre os protocolos mais utilizados estão o Wi-Fi, Bluetooth, ZigBee, GSM (Global System for Mobile Communication), RFID e LoRaWAN (ASPENCORE; 2019). Tendo em vista todo esse contexto de crescimento da internet e de dispositivos interconectados, houve também um grande aumento na geração de dados. Estima-se que até 2025 sejam gerados mais de 163 zettabytes (ZB) de dados no mundo (IDC; 2021).

Em aplicações IoT é muito comum de haver troca de mensagens entre diferentes dispositivos, o que é importante para que um dispositivo tenha conhecimento do estado do outro, para que assim possa efetuar uma ação de controle. O protocolo MQTT é muito utilizado para a troca de mensagens entre dispositivos IoT por ser leve e ter um bom desempenho em conexões com banda larga de baixa capacidade de tráfego de dados.

Apesar da escolha adequada de protocolos de comunicação, muitos projetos IoT falham devido a custos não previstos. Um fator importante nisso, e um dos principais contribuintes para o custo da aplicação está relacionado a quantidade de dados transferidos, um custo que pode ser parcialmente mitigado pela escolha de uma maneira eficiente de comunicação entre dispositivos e módulos (U-BLOX; 2020). Outro ponto importante para o sucesso de um projeto é o quesito de disponibilidade e tempo de resposta das aplicações, pois momentos de indisponibilidade ou alto tempo de resposta podem gerar prejuízos significativos a um negócio, por conta disto, ter uma arquitetura de software e hardware, que seja tolerante a falhas e tenha um tempo de resposta baixo é essencial (EATON, 2012).

1.1 OBJETIVO GERAL

Desenvolver uma arquitetura genérica para aplicações IoT resiliente e tolerante a falhas, com clientes implementando protocolo MQTT-NS e *gateways* distribuídos utilizando computação

de borda.

1.2 OBJETIVOS ESPECÍFICOS

1. Projetar uma arquitetura de IoT resiliente e tolerante a falhas;
2. Simular uma rede de sensores;
3. Implementar *gateways* distribuídos para a rede de sensores;
4. Balancear carga de clientes entre *gateways* quando houver indisponibilidade em um *gateway*;
5. Distribuir carga entre *brokers* durante alto *throughput* ou saturação em uma região;
6. Receber dados do *gateway* em um *broker* MQTT e enviar para nuvem;
7. Validar eficiência em ambientes catastróficos e críticos;

1.3 ESTRUTURA DO TRABALHO

Esse trabalho inicia com uma breve abordagem sobre o cenário de aplicações IoT na seção 2.1. Na sequência, os demais tópicos do capítulo 2 tratam sobre conceitos que envolvem uma aplicação IoT, passando por arquitetura, protocolos de comunicação, sistemas operacionais específicos para IoT e nuvem. Já na seção 3 é descrito a metodologia e materiais utilizados no trabalho. Por fim, na seção 4 e 5, temos, os resultados e conclusão.

2 REFERENCIAL TEÓRICO

2.1 INTERNET DAS COISAS

O termo Internet das coisas, mais conhecido como IoT que vem se popularizando cada vez mais nos últimos anos, foi proposto pelo britânico Kevin Ashton no ano de 1999, que utilizou o nome como título de uma apresentação a fim de chamar a atenção de executivos da empresa a Procter&Gamble (ASHTON, 2010).

IoT define uma rede de dispositivos físicos, softwares e outras tecnologias interconectados com o objetivo final de se comunicarem entre si e a Internet com a finalidade de automatizar, monitorar, controlar aplicações, produtos, locais, etc. IoT é, e ainda será utilizado em muitas áreas, como, por exemplo, eletrodomésticos, câmeras de vigilância, sensores de monitoramento, atuadores, displays, veículos, etc (ATZORI; IERA; MORABITO, 2010).

O crescimento do uso de dispositivos que utilizam Internet das coisas tem gerado grande impacto social e econômico no planeta, em uma análise do instituto global McKinsey, de 2015, analisando mais de 150 aplicativos específicos de IoT existentes que poderiam estar em uso generalizado em 10 anos, estimaram que eles poderiam ter um impacto econômico total de 3,9 a 11,1 trilhões de dólares por ano em 2025. (MANYIKA, 2015).

Uma das aplicações de IoT mais importantes são as cidades inteligentes. Para aumentar a eficiência, utilizam-se tecnologias de informação e comunicação, para trocar dados e para melhorar a qualidade dos serviços e o bem-estar dos usuários (PRADHAN, 2010).

Outro termo que surgiu com a explosão de aplicações IoT foi o de cidades inteligentes. Segundo Hammi *et al.* (2018) uma cidade inteligente é definida por arquitetura que interconecta estruturas físicas, tecnológicas, sociais e de negócios, tudo com objetivo de impulsionar o desenvolvimento coletivo. Também pode ser caracterizada por uma grande implantação de IoT, principalmente por aplicações M2M, que interconectam toda sua infraestrutura, formando um ecossistema inteligente e acessível por meio de diversas plataformas comuns aos habitantes. Um exemplo de aplicação a cidades inteligentes é o programa de Rede Elétrica Inteligente (REI), desenvolvido pela Copel no estado do Paraná, cujo objetivo é ter monitoria na rede de energia elétrica, onde o cliente pode acompanhar seu consumo de energia através de um aplicativo de celular e também toda vez que houver quedas de energia em uma localidade o medidor inteligente avisa a Copel imediatamente, que poder agir e encaminhar equipes para verificar a situação, tendo assim períodos menores de indisponibilidade aos seus clientes (COPEL; 2020).

Com sensores e dispositivos de IoT coletando informações em tempo real ou próximo a isso, um grande volume de dados é gerado. Normalmente esses dados são armazenados em algum local para serem posteriormente analisados. Devido ao grande volume, muitas vezes complexos, tem-se uma problemática para processar esses dados, com isso o termo *big data*, que visa tratar essa problemática, nasceu e se popularizou nos últimos anos.

2.1.1 Big Data

Além do crescimento no volume de dispositivos e aplicações IoT, teve-se uma crescente na geração de dados, pois quanto mais aplicações, mais dados de telemetria, monitoramento, etc, são gerados. Um conjunto de dados heterogêneos estruturados, ou não, que não conseguem ser processados por plataformas de processamento de dados é chamado de *Big Data*. Pode ter vários terabytes, petabytes ou até zetabytes. A quantidade de dados que define o *big data* hoje pode nunca atingir seus limites no futuro, pois as tecnologias de captura e armazenamento estão evoluindo. Além disso, um determinado volume de dados não estruturados pode ser denominado como *Big Data*, enquanto um mesmo volume de dados estruturados pode não ser definido como *Big Data*. Isso significa que a definição de *big data* vai além do domínio do volume de dados, depende também de características, como diversidade e velocidade (KAUR; SOOD, 2017).

Não basta somente armazenar os dados de forma estruturada ou não. Deixar esses dados parados é inútil, só se gera valor com dados a partir do momento que os mesmos são analisados e explorados em prol de um negócio, desbloqueando um grande potencial de tomadas de decisão, transformando um grande volume de dados em fatos e hipóteses (GANDOMI; HAIDER, 2015).

2.2 PROTOCOLOS DE COMUNICAÇÃO

Os protocolos de comunicação são um conjunto de regras que definem como as aplicações IoT implementam e padronizam a troca de informações entre si.

O protocolo determina como as mensagens trafegarão na rede, qual formato e método de propagação. Os elementos que compõem são a sintaxe, que define o formato e níveis de sinal; a semântica, que trata das informações de controle para o arranjo e tratativa de erros; por fim a temporização, que está relacionada a velocidade de seguimento dos dados (SILVA JUNIOR, 2021).

Com o surgimento dos protocolos de comunicação, surgiram alguns modelos de

referência para padronizar e garantir a comunicação entre sistemas. Dentre eles, um modelo de referência consolidado é o de camadas OSI, que se baseia em uma pilha protocolos independentes (TANENBAUM, 2003).

2.2.1 Modelo OSI de camadas

O modelo OSI do inglês (Open System Interconnection) é uma convenção para protocolos de rede, que visa definir as etapas e tarefas necessárias para conectar dois ou mais dispositivos. É definido por 7 camadas, ilustrado na Figura 1, onde cada camada pode ter várias subcamadas (LI *et al.*, 2011). Sendo definidas como:

- **Física** - A camada física define especificações elétricas e físicas dos dispositivos. Em especial, define a relação entre um dispositivo e um meio de transmissão, tal como um cabo de cobre ou um cabo de fibra óptica. Isso inclui o *layout* de pinos, tensões, impedância da linha, especificações do cabo, temporização, repetidores, adaptadores de rede e muito mais. É responsável por definir se a transmissão pode ser ou não realizada nos dois sentidos simultaneamente. Diz respeito a transmissão e recepção do fluxo de bits brutos não-estruturados em um meio físico. O fluxo de bits pode ser agrupado em palavras de código ou símbolos e convertido em um sinal físico que é transmitido através de um meio de transmissão de hardware (TANENBAUM, 2003).
- **Enlace de Dados** - A camada de enlace de dados usa os serviços da camada física abaixo dela para enviar e receber bits por canais de comunicação (possivelmente não confiáveis) que podem perder dados. Ele tem uma série de funções, incluindo:
  - Fornecimento de uma interface de serviço bem definida para a camada de rede
  - Enquadramento de sequências de bits/bytes
  - Detecção e correção de erros de transmissão
  - Regular o fluxo de dados para que os receptores lentos não sejam sobrecarregados por remetentes rápidos, entre outras funções.
- **Rede** - está relacionada a transmissão, nível de congestionamento da rede, qualidade de serviço e roteamento para determinar o custo e melhor caminho para trafegar de um ponto para outro. Quando um dado é enviado de uma origem a um destino, conforme a saturação da rede as mensagens podem ter uma priorização ou rota diferente, a camada de rede é responsável por gerenciar e controlar o roteamento entre a origem e destino do pacote. Dentre os protocolos da camada de rede temos: IP, IPX, RIP, OSPF,

etc.

- **Transporte** - responsável por fazer o transporte de dados entre máquinas, é independente de aplicação e rede físicas. Reúne protocolos de transporte M2M, dentre eles os mais utilizado são: TCP/IP, com garantia de entrega de mensagens; UDP mais rápido que TCP, porém sem garantia de entrega de mensagens. Para ocorrer o transporte dos dados, é essencial que as máquinas consigam se comunicar ente si. Protocolos da camada de transporte incluem: TCP, UDP, SPX.
- **Sessão** - se para o transporte de informações M2M, requer que as maquinas se comuniquem entre si, a camada de sessão é responsável por isso. Nesta etapa também pode ser feita verificação nos pontos para obter a sincronização de dados.Estabelecer links de comunicação durante a sessão, está as funções dessa camada. Basicamente a camada de sessão permite que diferentes máquinas estabeleçam sessões entre usuários.
- **Apresentação:** tradutor entre a rede de dados e o programa, nesta etapa, caso necessário há a conversão, compressão e criptografia caso necessite. Define uma série de códigos para garantir que a fonte de dados possa ser identificada e interpretada no destino.
- **Aplicação** - nível mais alto das aplicações de software, tem como principal função, fornecer interface para que os programas acessem serviços de rede. A camada de aplicação fornece serviços que incluem transferência de arquivos, gerenciamento de arquivos e processamento de informações por e-mail. Entre os protocolos presentes nessa camada estão: Telnet, FTP, HTTP, SNMP, MQTT.

2.3 MQTT

O protocolo MQTT, segundo a classificação do modelo OSI, funciona sob um protocolo de transporte subjacente, como o TCP, portanto fica na camada de aplicação. Criado em 1999 por Andy Stanford-Clack da IBM e Arlen Nipper da Arcom, e somente em 2013 foi padronizado no OASIS (LOCKE, 2010).

É um protocolo de transporte de mensagens de publicação/assinatura, onde se tem um ou mais elementos que publicam em um tópico e um ou mais elementos de interesse que se inscrevem para receber todas as publicações. É leve, simples e projetado para ser fácil de se utilizar. Essas características o tornam ideal para uso em muitas situações, incluindo aplicações restritas, com baixo poder de processamento e largura de banda limitada, sendo realidade na

Figura 1 – Camadas do Modelo OSI

Fonte: adapdato de TANENBAUM (2003).

grande maioria das aplicações IoT (GUPTA; 2018).

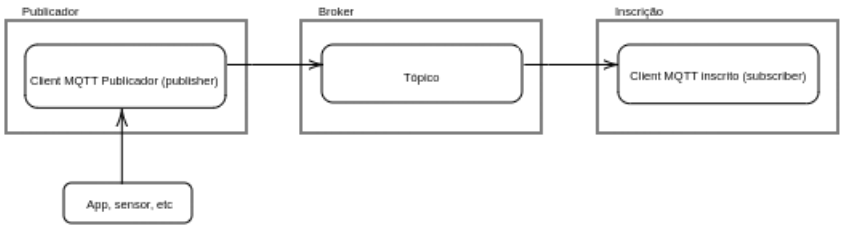
2.3.1 Arquitetura do protocolo

O padrão de publicação e assinatura do protocolo MQTT, também é conhecido como Pub/Sub, ilustrado na Figura 2. Nessa arquitetura é possível observar 3 componentes principais: Publicador (Publisher), Broker (Gerenciador intermediário) e Subscriber (Inscrição). Segundo Xu, Mahendran e Radhakrishnan (2016), cada componente é definido por:

- **Publisher** – dispositivos que geram dados. Os dados são publicados seguindo o formato de tópicos.
- **Broker** – O Intermediário age como um nó central realizando a comunicação entre Publicadores e Assinantes.
- **Subscriber** – Os Assinantes recebem as mensagens publicadas, conforme o tópico em que se inscreveram. Para publicar os dados por meio do MQTT, um *Publisher*, inicialmente realiza uma solicitação para conexão com o *Broker*.

Um publicador abre conexão com o *broker* e específica um tópico para fazer publicação de mensagens, por outro lado, temos um ou mais *subscriber* que abrem conexão com o *broker* e se inscrevem em tópicos de interesse, a partir do momento que um *subscriber* assina um tópico, irá receber todas as mensagens publicadas no mesmo. Os tipos de mensagens disponíveis tanto para publicador quanto para assinante são apresentadas na Tabela 1.

Figura 2 – Arquitetura MQTT



Fonte: Autoria própria (2022).

Tabela 1 – Mensagens protocolo MQTT

Valor	Nome	Direção	Descrição
0	Reservado	Proibido	Reservado
1	CONNECT	Cliente para Servidor	Requisição de cliente
2	CONNACK	Servidor para o Cliente	Reconhecimento de conexão
3	PUBLISH	Cliente para o Servidor	
4	PUBACK	Cliente para o Servidor	
5	PUBREC	Publicação recebida	Publicação recebida
6	PUBREL	Cliente para Servidor	
7	PUBCOMP	Cliente para Servidor	
8	SUBSCRIBE	Cliente para Servidor	Pedido de inscrição
9	SUBACK	Servidor para cliente	Reconhcimento de inscrição
10	UNSUBSCRIBE	Cliente para Servidor	Pedido de desinsção
11	UNSUBACK	Servidor para cliente	Reconhecimento desinscrição
12	PINGREQ	Requisição	Requisição PING
13	PINGRESP	Servidor para cliente	Resposta PING
14	DISCONNECT	Cliente para Servidor	Cliente esta desconectado
15	Reservado	Proibido	Reservado

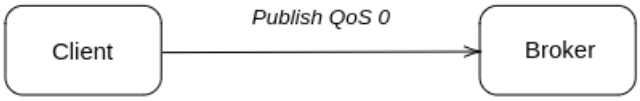
Fonte: Autoria própria (2022).

A conexão entre cliente e *broker* é criado utilizando protocolo TCP, tendo a opção de autenticação por meio de usuário e senha, e com criptografia (SSL / TLS). Também é possível estabelecer conexão de outras formas, como, por exemplo, MQTT rodando sob links seriais, por exemplo. (BARROS, 2015)

Cada processo de conexão também estabelece um nível de qualidade de serviço desejado (QoS, Quality of Service), apontando como deve ser a relação entre os elementos que se comunicam. Existem três níveis de qualidade de serviço:

- **QoS 0 (*No máximo uma vez*)** - É semelhante ao protocolo de transporte UDP, onde não possui uma confirmação de mensagem, o cliente somente envia a mensagem ao *broker*, conforme ilustrado na Figura 3. Os remetentes não precisam armazenar mensagens para retransmissões futuras.
- **QoS 1 (*Pelo menos uma vez*)** - Há a confirmação de entrega de uma mensagem,

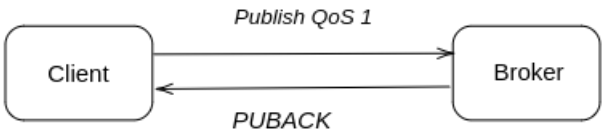
Figura 3 – QoS 0



Fonte: adapdato de HIVEMQ (2015).

devido o remetente enviar várias mensagens análogas, ilustrado na Figura 4. Com isso pode ocorrer um atraso no recebimento de feedback, neste caso, é garantido, que uma das mensagens terá o reconhecido bem-sucedido. Para isso a mensagem é armazenada pelo remetente até confirmação de recebimento;

Figura 4 – QoS 1

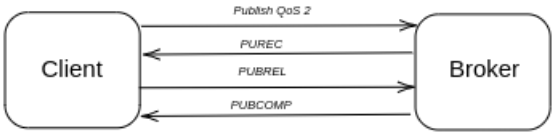


Fonte: adapdato de HIVEMQ (2015).

- **QoS 2 (*Exatamente uma vez*)** - Garante que a mensagem só será entregue uma única vez, com envio de confirmações de recebimento e confirmações de recebimento de confirmações de recebimento, se tem uma confirmação bidirecional conforme ilustra a Figura 5. Similar a confirmação do protocolo de TCP.

Não existe um QoS melhor ou pior, tudo depende do caso de uso da aplicação, recursos disponíveis, etc. O nível de QoS é acordado entre *client* e *broker*, logo em um mesmo tópico podemos ter níveis de QoS heterogênios (BARROS, 2015). Embora o broker represente um

Figura 5 – QoS 2



Fonte: adapdato de HIVEMQ (2015).

ponto de falha na rede ao centralizar as comunicações, ele permite o desacoplamento entre as partes comunicantes, o que não é possível nos modelos de comunicação cliente/servidor. Uma saída para mitigar esse ponto de falha é usar *brokers* clusterizados com várias instâncias rodando em diferentes máquinas e regiões (BARROS, 2015).

2.4 REDES DE SENSORES

Uma rede de sensores consiste em um grande número dispositivos sensores ou atuadores operados por bateria, sendo geralmente equipados com uma quantidade limitada de recursos de armazenamento e processamento. As Redes de Sensores Sem Fio (WSN) podem servir a diferentes propósitos, desde medição e detecção até automação e controle de processos (STANFORD-CLARK; TRUONG, 2013).

2.4.1 MQTT-SN

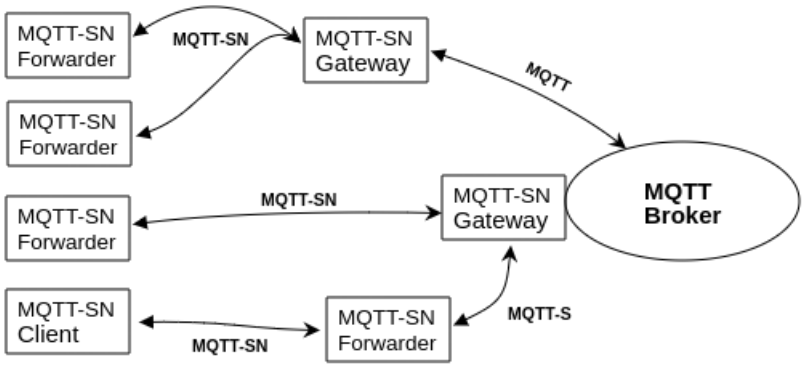
O protocolo MQTT é otimizado para comunicação em redes em que a largura de banda é limitada. O ponto negativo é que por operar sob o protocolo TCP/IP, necessita de conexão ordenada sem perdas, isso é complexo para dispositivos simples e limitados. O MQTT-SN pode ser considerado como uma versão do MQTT adaptada às peculiaridades de um ambiente de comunicação sem fio. Originalmente foi feito para funcionar com ZigBee, mas pode operar sob qualquer protocolo de rede que tenha um serviço de transferência de dados bidirecional entre um cliente e um gateway (STANFORD-CLARK; TRUONG, 2013).

2.4.1.1 Arquitetura MQTT-SN

Na Figura 6 pode-se observar que na arquitetura do MQTT-SN existem 3 componentes principais, que Stanford-Clark e Truong (2013) definem como:

- **Clients** - Dispositivos finais da rede de sensores, podem publicar ou receber dados dos gateways MQTT-SN ou *Forwarder* por meio do protocolo MQTT-SN;
- **Gateways** - Faz a tradução de dados entre clientes MQTT-SN ou *Forwarder* para *brokers* MQTT. Responsável pela tradução entre o protocolo MQTT e MQTT-SN;
- **Forwarder** - Caso o *Gateway* não esteja na mesma rede do cliente, pode-se utilizar um *forwarder* para encaminhar e receber dados entre cliente e *gateway*. Encapsula os quadros MQTT-SN que recebe do cliente e os encaminha sem alteração para o *gateway*. Também recebe dados do *getway* encapsulado, desencapsula os quadros e os envia aos clientes sem alteração;

Figura 6 – Arquitetura MQTT-SN



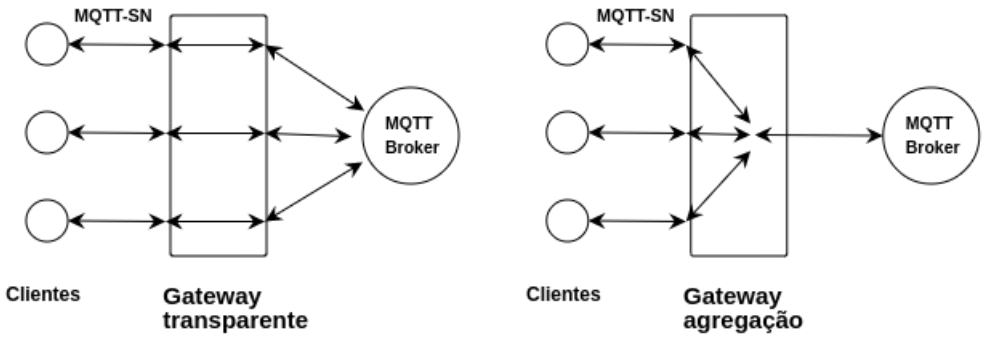
Fonte: adaptado de Stanford-Clark e Truong (2013).

Conforme os dados são traduzidos pelo gateway entre MQTT e MQTT-SN, como ilustrado pela Figura 7, pode-se classificar os gateways em dois tipos (STANFORD-CLARK; TRUONG, 2013):

- **Gateway transparente** - Tem sua implementação mais simples, pois, mantém conexão de cada cliente MQTT-SN com o *broker* MQTT, a troca de mensagens é mantida o mais próximo ao original, a única mudança feita pelo gateway é a tradução de sintaxe entre os dois protocolos. Todas as funções e funcionalidades implementadas pelo servidor podem ser oferecidas ao cliente

- **Gateway de agregação** - Mantém somente uma conexão MQTT com o *broker* e decide quais informações são fornecidas. Tem implementação mais complexa, porém, pode manter um número maior de dispositivos conectados em uma rede de sensores, pois, reduz o número de conexões com o *broker*.

Figura 7 – Gateway transparente e de agregação



Fonte: adaptado de Stanford-Clark e Truong (2013).

Para a implementação do protocolo MQTT-SN, como mencionado anteriormente, necessitam-se outros protocolos de comunicação e, em geral, de um sistema operacional. Alguns sistemas operacionais foram criados exclusivamente para redes de sensores, como, o RIOT-OS que será abordado na sequência.

2.5 SISTEMA OPERACIONAL RIOT-OS

Segundo Bormann, Ersue e Keranen (2014), dispositivos de IoT costumam ter recursos de hardware limitados, no geral dispositivos de baixo custo não tem poder computacional para executar SOs complexos como Linux ou Windows. Assim surge a necessidade de sistemas operacionais dedicados para IoT, mais enxutos e com recursos nativos de melhor desempenho que também facilitam na hora do desenvolvimento de aplicações *IoT*. O sistema operacional RIOT é um exemplo de SO desenvolvido especialmente para aplicações IoT, baseado em uma arquitetura modular construída em torno de um kernel minimalista e desenvolvido por uma comunidade mundial de desenvolvedores, escrito para dispositivos de baixo custo. Para sua execução exige memória mínima de 10KB e não necessita que o hardware possua MMU nem MPU (BACCELLI *et al.*, 2018). Além de necessitar menos memória, o RIOT suporta uma gama grande de arquiteturas de 8 a 32 bits. Também fornece conjunto completo de recursos de um

sistema operacional, desde abstração de hardware, recursos de kernel, bibliotecas de sistema até ferramentas.

Alguns princípios que explicam partes do projeto RIOT são:

- **Padrões de rede:** Se concentra em especificações de protocolo de rede padrão aberto, por exemplo, protocolos IETF.
- **Padrões do sistema:** Cumpre os padrões relevantes, por exemplo, o padrão ANSI C (C99), para aproveitar ao máximo o maior conjunto de softwares de terceiros. O uso da linguagem C atende a requisitos de recursos de baixo nível e fácil programação.
- **APIs unificadas:** Fornece consistência de APIs em todos os hardwares suportados, mesmo para APIs de acesso a hardware, para atender à portabilidade de código e minimizar a duplicação de código.
- **Modularidade:** Define blocos de construção independentes, a serem combinados de todas as maneiras possíveis, para atender a casos de uso versáteis e restrições de memória.
- **Memória estática** - Faz uso extensivo de estruturas pré-alocadas para atender tanto à confiabilidade, validação e verificação simplificadas, quanto aos requisitos em tempo real.
- **Independência de fornecedor e tecnologia** - As bibliotecas de fornecedor são normalmente evitadas, para evitar o aprisionamento do fornecedor e para minimizar a duplicação de código, além disso, as decisões de design não devem vincular a RIOT a uma tecnologia específica.
- **Comunidade de código aberto aberta e inclusiva** - Permanece livre e aberto para todos e agregar uma comunidade com processos 100% transparentes.

2.6 ARQUITETURA DE ALTA DISPONIBILIDADE

Com a junção de componentes tanto da camada física como processadores e discos, quanto da camada de software, como protocolos de comunicação e SOs, temos uma arquitetura, que seria a união de todos esses componentes trabalhando de forma cooperativa. Uma arquitetura de alta disponibilidade é o conjunto de tecnologias em que se tem o mínimo de interrupções e indisponibilidade de serviços de computação, proporcionando serviços redundantes e tolerantes a falhas em um data center.

Para a maioria das aplicações de IoT, a alta disponibilidade é fundamental. Existem

várias formas de atingir essa disponibilidade em um sistema, por exemplo, detecção e recuperação de falhas (YANG; KIM, 2019).

- **Detecção de falhas** - Pode ser feita ao nível físico, virtual e de aplicação, uma aplicação pode conter um ou mais níveis.
- **Recuperação de falhas** - Podem ser classificados como tolerância a falhas e métodos de redundância, os métodos de tolerância a falhas visa estabilizar o sistema reiniciando o sistema ou redirecionando o fluxo para outro servidor normal. Já os métodos de redundância melhoram um sistema preparando um sistema igual ao do serviço prestado e utilizando-o como substituto em caso de falha. Os métodos de redundância geralmente são mais rápidos que os de tolerância a falhas, devido a sempre ter o mesmo sistema em *backup* ou redundante, mas, por outro lado, são mais custosos de manter.

2.7 MODELOS DE COMPUTAÇÃO

Quando se fala de modelos de computação ou formas de implantar aplicações, tem-se três formas:

- **Nuvem** - Empresa contrata serviços de computação de terceiros para manter suas aplicações e não tem responsabilidade sob manutenção e atualizações dos servidores.
- **On-premise** - Empresa mantém seus servidores em sua estrutura física e responsável por todo o processo de *upgrade* e manutenção.
- **Híbrido** - Empresa mantém parte das aplicações em sua estrutura e outra parte na nuvem.

Nos últimos anos o modelo de computação de nuvem entrou em ascensão por muitos motivos, tais como menor custo de operação, não necessitar de time especializado dedicado a cuidar de servidores, economia de espaço e recursos energéticos. Outro ponto muito importante é a elasticidade provida pelo modelo de computação em nuvem, em que é muito fácil redimensionar memória e processamento nos recursos de computação.

2.7.1 Computação em nuvem

Com a expansão tecnológica em todos os setores, se deu uma alta demanda por processamento em nuvem, muito devido a aplicações terem mudanças de dimensão muito

rápidas, a elasticidade provida pela computação em nuvem ajuda empresas a dimensionar o poder computacional necessário para suas soluções de forma simples e rápida. No entanto, enquanto as tecnologias de nuvem trazem recursos e serviços de alto desempenho para empresas com melhor disponibilidade e preço, muitos aplicativos exigem uma forte necessidade de elasticidade, escalabilidade de recursos e baixa latência na transmissão de dados (BAKHOUYA *et al.*, 2020).

2.7.1.1 Arquitetura serverless

No modelo tradicional de IasS se tem servidores com capacidade provisionada executando a todo momento, o que é um desperdício computacional para determinadas aplicações, principalmente quando não se tem previsibilidade ou constância do uso. Já em computação serverless, tem-se execução de trechos de código sob demanda com base em eventos, onde recursos são alocados para processar determinado evento, e se é pago somente pelos recursos utilizados durante o período de execução. Toda a parte de infraestrutura e dimensionamento de serviços de tecnologia de informação fica sob responsabilidade do provedor dos serviços, sobrando assim mais tempo para as empresas se preocuparem em desenvolver suas soluções.

2.7.2 Computação de borda

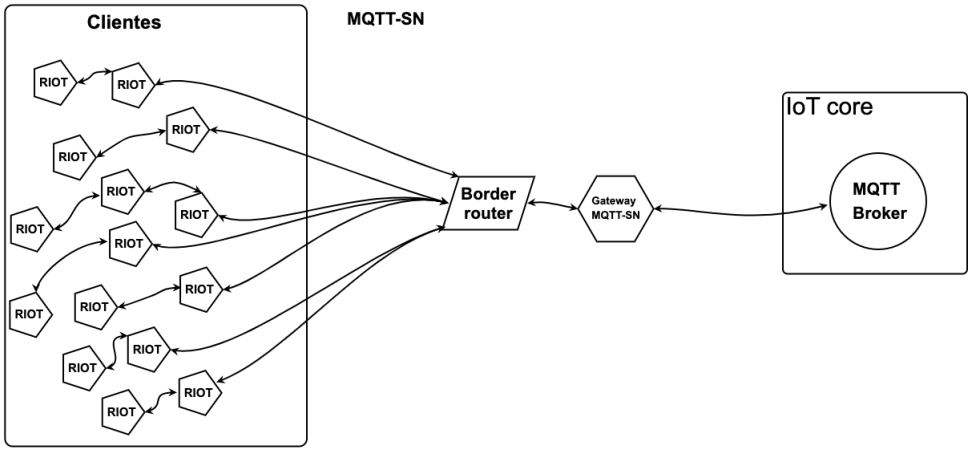
Computação de borda é o nome dado a processos de computação que são executados próximos à fonte de dados Cao *et al.* (2020), os dados são coletados por dispositivos e passam por um processamento próximo a si, antes de serem enviados a nuvem.

Para Satyanarayanan (2017), professor da Universidade Carnegie Mellon, computação de borda é definido como a execução de recursos na borda da rede mais próxima de dispositivos ou sensores móveis. É um novo modelo de computação que une recursos que estão próximos ao usuário em distância geográfica ou distância de rede para fornecer serviços de computação, armazenamento e rede para aplicativos.

3 METODOLOGIA DE DESENVOLVIMENTO

A Figura 8 mostra uma rede de sensores convencional onde se tem apenas um *border router* e um *gateway* MQTT-SN. Essa arquitetura é adequada, porém tem um ponto de falha único, basta ocorrer um problema no *border router* ou no *gateway* e todo o sistema falha.

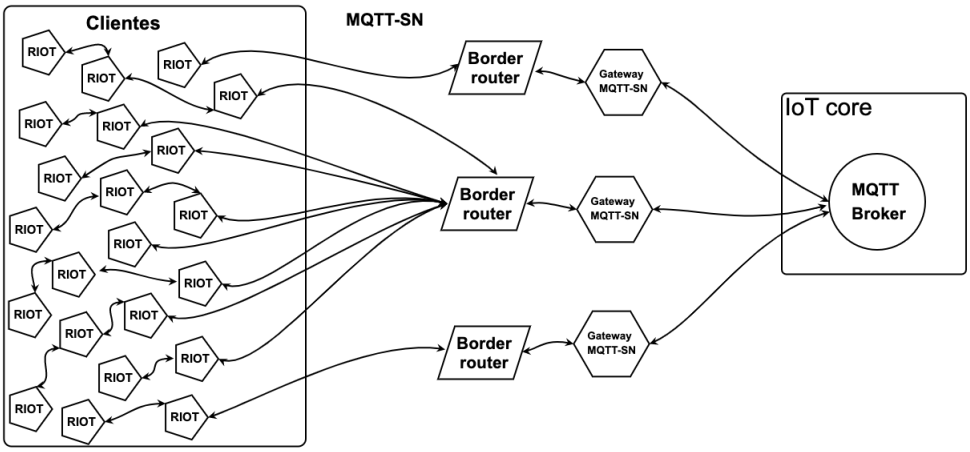
Figura 8 – Rede de sensores convencional



Fonte: Autoria própria (2022).

Para tratar essa problemática de disponibilidade é proposto a implantação de uma arquitetura multi border router e multi-gateway conforme a Figura 9. Ao ter indisponibilidade em um border router, o sistema tem a resiliência de redirecionar o tráfego de dados para o border router disponível mais próximo, tendo assim uma garantia de disponibilidade.

Figura 9 – Rede de sensores multi-gateway



Fonte: Autoria própria (2022).

Este capítulo apresenta a metodologia de desenvolvimento do trabalho proposto, iniciando com a configuração do ambiente de desenvolvimento. Posteriormente, com o ambiente

configurado, inicia-se o desenvolvimento de firmware utilizando o sistema operacional RIOT para os dispositivos presentes na rede de sensores simulada. Com o *firmware* desenvolvido se dá início a simulação da rede de sensores em que são utilizados *gateways* para o protocolo MQTT-SN fazendo a ponte com o protocolo MQTT. Após ter uma rede de sensores funcionando e comunicando com um broker MQTT, é simulado cenários de desbalanceamento de carga entre clientes e broker MQTT de modo a configurar balanceamento de carga para esses cenários. Por fim se tem o envio dos dados do broker para a nuvem e validação da arquitetura em sua totalidade.

3.1 PREPARAÇÃO DO AMBIENTE DE DESENVOLVIMENTO

Para começar a desenvolver a arquitetura proposta é necessário preparar o ambiente de desenvolvimento. Para isso um conjunto de ferramentas serão configuradas e instaladas. As ferramentas utilizadas são plataformas hospedadas em nuvem e softwares de código aberto, dentre elas:

- FIT IoT-LAB;
- AWS IoT core;
- Visual Studio Code v1.73;
- Ubuntu v20.04.2 LTS;
- Docker v20.10.21;
- Docker Compose v3.0;
- MQTT-SN v1;
- MQTT v3.1.1;
- RIOT-OS 2022.07;

3.2 DESENVOLVIMENTO FIRMWARE RIOT-OS

Após instalar todos os softwares listados acima se dá início do desenvolvimento do firmware para a aplicação IoT, para isso será utilizado o sistema operacional RIOT-OS e a linguagem de programação C. Por meio de um algoritmo pseudo randômico a aplicação desenvolvida tem como intuito gerar dados fictícios de temperatura, umidade e pressão atmosférica e enviá-los para um gateway via protocolo MQTT-SN. Tal envio irá ser realizado periodicamente em prazos de poucos segundos. Os dados recebidos pelo gateway serão tratados e encaminhados para um broker MQTT. O firmware desenvolvido será implantado em vários dispositivos em um

simulador de rede de sensores.

3.3 SIMULAÇÃO REDE DE SENSORES

A simulação dos dispositivos e da rede de sensores foi realizada através do FIT IoT-LAB, uma plataforma para testes de redes de dispositivos e sensores sem fio heterogêneos, com isso é possível programar os dispositivos e controlá-los. Essa plataforma é utilizada por grande parte da comunidade de IoT pelo mundo (IOT-LAB, 2022).

O primeiro passo para a simulação de um novo experimento nessa plataforma é o upload de um firmware previamente desenvolvido. Após o upload, deve-se selecionar em qual hardware o firmware será implantado, quantas instâncias serão utilizadas, por quanto tempo e em qual localidade deseja-se executar os dispositivos.

3.4 CONFIGURAÇÃO DE GATEWAYS MQTT-SN PARA REDE DE SENSORES

O *gateway* MQTT-SN utilizado para o projeto será o Mosquitto Really Small Message Broker (RSMB) por ser uma implementação de servidor dos protocolos MQTT e MQTT-SN. Qualquer cliente que implemente este protocolo corretamente pode usar este servidor para enviar e receber mensagens, o qual tem como finalidade receber os dados da rede de sensores e fazer a ponte entre o protocolo MQTT-SN e MQTT.

3.5 CONFIGURAÇÃO BROKER MQTT

Para o protocolo MQTT será utilizado o recurso da AWS IoT core, que prevê um broker que roda em cima de uma arquitetura serverless e ser totalmente gerenciado pela AWS a qual gerante uma alta disponibilidade e API simplificada para comunicação através da sua SDK. Por não ser possível fazer o envio de dados do gateway MQTT-SN diretamente para o AWS IoT core, será necessário desenvolver um *script* para fazer essa ponte entre o *gateway* e a AWS. A linguagem que será utilizada para o *script* será a Golang por ser compilada e ter um bom desempenho para concorrência e paralelismo. Também será utilizado a SDK da AWS para se comunicar com o serviço IoT core de forma simples, necessitando somente configurar credenciais da AWS e chamar algumas funções pré-implementadas para envio de dados via MQTT.

3.6 VALIDAÇÃO DA ARQUITETURA

Para validar a arquitetura serão realizados vários testes de caso, começando por envio de dados simples dos dispositivos da rede de sensores via gateway chegando até o broker MQTT. Após isso serão realizados testes em cenários com indisponibilidade de um border router e gateway MQTT-SN, visando que o sistema faça um rebalanceamento do tráfego do gateway indisponível para um novo gateway. Por último, o teste de saturação da rede, em que se tem um gateway sobrecarregado e a rede de sensores deve redirecionar parte do tráfego para outro border router próximo.

3.7 CRONOGRAMA

Atividades	Ago/22	Set/22	Out/22	Nov/22	Dez/22	Jan/23	Fev/23	Mar/23	Abr/23	Maio/23	Jun/23	Jul/23
Revisão bibliográfica	X	X	X	X	X	X	X	X	X	X	X	
Configuração de ambiente				X	X	X						
Simulação rede de sensores					X	X	X	X				
Implementação arquitetura multi-gateway						X	X	X	X			
Resultados parciais										X		
Testes em cenários catastróficos									X	X		
Coleta métricas											X	
Apresentação de resultados												X

Tabela 2 – Cronograma do Projeto.

REFERÊNCIAS

ASHTON, K. **That Ïnternet of ThingsThing**. [*S. l.: s. n.*], 2010. RFID Journal. Internet of Things. Disponível em: http://www.rfidjournal.com/articles/view?4986. Acesso em: 15 set. 2022.

ASPENCORE; **Integrating IoT and Advanced Technology Designs, Application Development Processing Environments**. [*S. l.: s. n.*], 2019. Embedded Markets Study. Internet of Things. Disponível em: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide. Acesso em: 15 set. 2022.

ATZORI, L.; IERA, A.; MORABITO, G. The Internet of Things: A Survey. **Computer Networks**, p. 2787–2805, out. 2010. DOI: 10.1016/j.comnet.2010.05.010.

BACCELLI, E. *et al.* RIOT: An open source operating system for low-end embedded devices in the IoT. **IEEE Internet of Things Journal**, IEEE, v. 5, n. 6, p. 4428–4440, 2018.

BAKHOUYA, M. *et al.* Cloud computing, IoT, and big data: Technologies and applications. **Concurrency and Computation: Practice and Experience**, v. 32, n. 17, e5896, 2020. DOI: https://doi.org/10.1002/cpe.5896. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.5896. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5896.

BARROS, M. **MQTT Publish/Subscriber - Protocolos para IoT**. [*S. l.: s. n.*], 2015. Embarcados - Sua fonte de informações sobre Sistemas Embarcados. Internet of Things. Disponível em: https://embarcados.com.br/mqtt-protocolos-para-iot. Acesso em: 20 set. 2022.

BORMANN, C.; ERSUE, M.; KERANEN, A. **Terminology for constrained-node networks**. [*S. l.*], 2014.

CAO, K. *et al.* An Overview on Edge Computing Research. **IEEE Access**, v. 8, p. 85714–85728, 2020. DOI: 10.1109/ACCESS.2020.2991734.

COPEL; **Rede Elétrica Inteligente**. [*S. l.: s. n.*], 2020. copel. Rede Elétrica Inteligente. Disponível em: https://www.copel.com/site/copel-distribuicao/rede-eletrica-inteligente/. Acesso em: 12 dez. 2022.

EATON, K. **How One Second Could Cost Amazon $1.6 Billion In Sales**. [*S. l.: s. n.*], 2012. Fast Company. Sales. Disponível em: https://www.fastcompany.com/1825005/how-one-second-could-cost-amazon-16-billion-sales. Acesso em: 20 set. 2022.

GANDOMI, A.; HAIDER, M. Beyond the hype: Big data concepts, methods, and analytics. **International Journal of Information Management**, v. 35, n. 2, p. 137–144, 2015. ISSN 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2014.10.007. Disponível em: https://www.sciencedirect.com/science/article/pii/S0268401214001066.

GUPTA; A. B. E. B. K. B. R. **MQTT Version 5.0**. 02. ed. [*S. l.*], 2018.

HAMMI, B. *et al.* IoT technologiesfor smart cities. **IET Networks**, v. 7, n. 1, p. 1–13, 2018. DOI: https://doi.org/10.1049/iet-net.2017.0163. eprint: https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-net.2017.0163. Disponível em: https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/iet-net.2017.0163.

HIVEMQ, T. **Quality of Service (QoS) 0,1**. [*S. l.: s. n.*], 2015. hivemq. MQTT Essentials: Part 6. Disponível em: https://www.hivemq.com/blog/mqtt-essentials-part-6-mqtt-quality-of-service-levels. Acesso em: 22 set. 2022.

IDC; **Data Creation and Replication Will Grow at a Faster Rate than Installed Storage Capacity, According to the IDC Global DataSphere and StorageSphere Forecasts**. [*S. l.: s. n.*], 2021. IDC: The premier global market intelligence company. Internet of Things. Disponível em: %3Chttps://www.idc.com/getdoc.jsp?containerId=prUS47560321. Acesso em: 15 set. 2022.

IOT-LAB. **FIT IoT-LAB**. [*S. l.: s. n.*], 2022. FIT IoT-LAB. FIT IoT-LAB. Disponível em: https://www.iot-lab.info. Acesso em: 3 dez. 2022.

KAUR, N.; SOOD, S. K. Dynamic resource allocation for big data streams based on data characteristics (5Vs). **International Journal of Network Management**, v. 27, n. 4, e1978, 2017. e1978 NEM-16-0189.R2. DOI: https://doi.org/10.1002/nem.1978. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/nem.1978. Disponível em: https://onlinelibrary.wiley.com/doi/abs/10.1002/nem.1978.

LI, Y. *et al.* Research based on OSI model. In: 2011 IEEE 3rd International Conference on Communication Software and Networks. [*S. l.: s. n.*], 2011. P. 554–557. DOI: 10.1109/ICCSN.2011.6014631.

LOCKE, D. **Mqtt v3. 1 protocol specification. International Business Machines**. 3. ed. [*S. l.*], 2010. p. 42.

MANYIKA, J. **By 2025, Internet of things applications could have $11 trillion impact**. [*S. l.: s. n.*], 2015. McKinsey & Company. Internet of Things. Disponível em: https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact. Acesso em: 15 set. 2022.

PRADHAN, B. **IoT-Based Applications in Healthcare Devices**. [*S. l.*]: Journal of healthcare engineering, 2010. 2166325998 p. (2021). ISBN 9780451012821.

SATYANARAYANAN, M. The emergence of edge computing. **Computer**, IEEE, v. 50, n. 1, p. 30–39, 2017.

SILVA JUNIOR, M. P. d. **Análise entre protocolos HTTP e MQTT em projetos IOT**. Nov. 2021. Monografia (TCC) – Universidade Tecnológica Federal do Paraná.

STANFORD-CLARK, A.; TRUONG, H. L. Mqtt for sensor networks (mqtt-sn) protocol specification. **International business machines (IBM) Corporation version**, v. 1, n. 2, p. 1–28, 2013.

TANENBAUM, A. S. **Redes de Computadores: Tradução: Vandenberg D. de Souza**. [*S. l.*]: Rio de Janeiro: Elsevier, 2003.

U-BLOX; **MQTT-SN – lowering the cost of IoT at scale**. [*S. l.: s. n.*], 2020. u-blox. Internet of Things. Disponível em: https://www.u-blox.com/en/blogs/insights/mqtt-sn#:~:text=MQTT%2DSN%20. Acesso em: 20 set. 2022.

VAILSHERY, L. S. **IoT devices installed base worldwide 2015-2025**. [*S. l.: s. n.*], 2016. Statista. Internet of Things. Disponível em: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide. Acesso em: 15 set. 2022.

YANG, H.; KIM, Y. Design and Implementation of High-Availability Architecture for IoT-Cloud Services. **Sensors**, v. 19, n. 15, 2019. ISSN 1424-8220. DOI: 10.3390/s19153276. Disponível em: https://www.mdpi.com/1424-8220/19/15/3276.