Tatiana Reimer Barata

Uma extensão do software Mobile Hub para torná-lo multiprotocolo

RELATÓRIO DE PROJETO FINAL

DEPARTAMENTO DE ENGENHARIA ELÉTRICA E DEPARTAMENTO DE INFORMÁTICA

Programa de graduação em Engenharia de Computação

Rio de Janeiro
Junho de 2023

Pontifícia Universidade Católica
do Rio de Janeiro

Tatiana Reimer Barata

Uma extensão do software Mobile Hub para torná-lo multiprotocolo

Relatório de Projeto Final

Relatório de Projeto Final, apresentado ao programa de Engenharia de Computação da PUC-Rio como requisito parcial para a obtenção do titulo de Engenheiro de Computação.

Orientador : Prof. Markus Endler
Co-orientador: Luiz Eduardo Talavera

Rio de Janeiro
Junho de 2023

Agradecimentos

Ao meu orientador Markus Endler e ao meu co-orientador Luiz Eduardo Talavera pela oportunidade de participar desse projeto.

À Millena Cavalcanti, que me acompanhou durante o projeto final II.

À minha família, que sempre me apoiou.

Ao meu namorado Vinícius, por ter ouvido todas as minhas reclamações.

A todos os amigos que fiz ao longo desses 6 anos.

À PUC-Rio.

Resumo

Reimer Barata, Tatiana; Endler, Markus; Talavera, Luiz Eduardo. **Uma extensão do software Mobile Hub para torná-lo multiprotocolo**. Rio de Janeiro, 2023. 42p. Projeto de Graduação – Departamento de Engenharia Elétrica e Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Na versão atual, o middleware Mobile Hub (M-Hub) se comunica com dispositivos periféricos apenas por Bluetooth Low Energy. Com o crescimento do número de dispositivos IoT de forma heterogênea, existem diversos protocolos sendo utilizados, tornando necessário que o M-Hub tenha a capacidade de ser multiprotocolo. Este projeto irá estender o Mobile Hub com protocolos alternativos.

Palavras-chave

Internet das Coisas; Internet das Coisas Móveis; IoT; IoMT; Middleware; Sistemas Distribuídos

Abstract

Reimer Barata, Tatiana; Endler, Markus (Advisor); Talavera, Luiz Eduardo (Co-Advisor). **An extension of the Mobile Hub software to enable multiprotocol**. Rio de Janeiro, 2023. 42p. Final Project – Departamento de Engenharia Elétrica e Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In the latest version, the Mobile Hub middleware (M-Hub) communicates with peripheral devices only through Bluetooth Low Energy. With the increasing number of heterogeneous IoT devices, there are various protocols being used, making it necessary for M-Hub to have multiprotocol capability. This project will extend the Mobile Hub with alternative protocols.

Keywords

Internet of Things; Internet of Mobile Things; IoT; IoMT; Middleware; Distributed Systems

Sumário

Lista de figuras

Lista de tabelas

1
Introdução

A Internet das Coisas (IoT) descreve uma rede de objetos físicos incorporados a sensores, software e outras tecnologias com objetivo de se conectarem e transferir dados entre si e com outros sistemas pela internet (IBM 2016). Com as altas taxas de crescimento do número de dispositivos IoT — e a tendência de criação e utilização de dispositivos móveis, sejam eles smartphones ou sensores e atuadores embarcados, nestas redes — a infraestrutura de conectividade tem evoluído para redes heterogêneas, onde tecnologias de comunicação sem fio, cobertura, parâmetros de configuração, de comunicação e de segurança, taxa de dados, latência de transmissão, entre outros elementos, são distintas. A utilização crescente destes dispositivos inteligentes com mobilidade, como smartphones, drones e carros, remete a uma nova organização denominada Internet das Coisas Móveis (IoMT).

As aplicações da IoMT são diversas, podendo ser usadas em sistemas de monitoramento no meio ambiente, em cidades e casas inteligentes, no monitoramento em hospitais, no gerenciamento de energia, entre outros (Talavera 2015). Nesse contexto, a mobilidade impõe mais desafios com relação à descoberta de dispositivos, pois com um alto número de conexões e desconexões, torna-se importante lidar com a descoberta de dispositivos, com o processo de *handover*, que é a mudança do ponto de conexão enquanto a comunicação ainda está acontecendo.

Por exemplo, em cidades inteligentes, ônibus que circulam pela cidade podem coletar dados de temperatura de regiões da cidade utilizando sensores de temperatura e Sistema de Posicionamento Global (GPS). Nesse caso, os sensores dos ônibus precisam se conectar e desconectar rapidamente com os gateways. Já em hospitais, profissionais da área da saúde podem coletar dados dos sensores usados pelos pacientes enquanto eles percorrem o hospital.

O *middleware* Mobile Hub (M-Hub) (Talavera 2015), desenvolvido pelo Laboratory for Advanced Collaboration (LAC), propõe que smartphones se transformem em nós propagadores para os dispositivos mais simples que não tem acesso à internet, chamados de M-OBJs, e fornece informação de contexto dos dispositivos que estão conectados ao ContextNet Core, que suporta monitoramento em tempo real, unicast, groupcast e broadcast para

nós móveis.

Em sua versão mais recente, o M-Hub utiliza os protocolos Message Queuing Telemetry Transport (MQTT) e Mobile Reliable User Datagram Protocol (MR-UDP) para conexão com o Gateway, e o protocolo Bluetooth Low Energy (BLE) para conexão com os dispositivos periféricos.

Com o crescimento heterogêneo do número de dispositivos disponíveis, apesar de existir padronizações de comunicação, diversos protocolos de comunicação, como Bluetooth, BLE, ZigBee, Wi-Fi Direct, IPv6 over Low power Wireless Personal Area Networks (6LoWPAN), Near Field Communication (NFC) etc. são utilizados. Assim, para abranger um grande número de dispositivos, é preciso que o M-Hub tenha a capacidade de ser multiprotocolo.

Nesse sentido, o Mobile Hub será estendido com os protocolos WPAN Wi-Fi Direct e NFC. Ambos possuem características diferentes do BLE em relação às distâncias de comunicação e à latência, atribuindo ao M-Hub novas funcionalidades.

Uma aplicação para o Wi-Fi Direct estaria relacionada à comunicação entre M-Hubs. Por exemplo, um M-Hub, chamado M-Hub_a, tem acesso a alguns periféricos BLE e envia dados para o ContextNet Core usando WLAN. A 150 metros dele existem outros sensores BLE próximos de outro M-Hub, chamado M-Hub_b. Em algum momento, o M-Hub_b fica sem contectividade com o ContextNet Core. Então, o M-Hub_a pode se tornar um Wi-Fi Direct Group Owner e fornecer conexão para o M-Hub_b, que passa a ser seu cliente. Após estabelecida a conexão, o M-Hub_b de tempos em tempos pode encaminhar seus dados coletados dos sensores em seu entorno (em lote) e enviá-los para o M-Hub_a. Ao receber os dados, o M-Hub_a pode processá-los localmente comparando-os com os dados do sensor ao seu redor, ou enviá-los para o ContextNet Core.

Para o NFC, considera-se um cenário onde todas as salas de aula do Ed. Leme possuem mini-displays eletrônicos em suas portas, exibindo a programação de aulas ou atividades durante o dia em cada sala. Supondo que durante um grande evento, várias salas do edifício precisariam ter sua programação alterada. Essa reprogramação poderia ser feita de forma centralizada, e deixada disponível em um servidor *backend* no ContextNet Core. Então, bastaria que um funcionário percorresse as portas das salas, encostasse o seu M-Hub em cada um dos mini-displays eletrônicos e, através do NFC, carregasse a nova programação específica que cada sala terá no evento.

1.1 Objetivo

O objetivo deste projeto é estender o Mobile Hub com os protocolos WPAN Wi-Fi Direct e NFC, permitindo uma comunicação multiprotocolo, ampliando o potencial número de dispositivos distintos se comunicarem através do Mobile Hub.

A fim de permitir o suporte do Mobile Hub a multiprotocolos, alguns objetivos específicos serão necessários:

- Criar módulos para os protocolos de comunicação Wifi-Direct e NFC independentes que podem ser adicionados no código atual sem precisar alterá-lo.
- Simular cenários em que BLE, Wi-Fi Direct e NFC são utilizados demonstrando suas diferenças quanto a descoberta de dispositivos, conexão, leitura de dados e desconexão.

2
Fundamentos

Para o desenvolvimento de módulos no Mobile Hub é essencial entender todo o seu funcionamento e o sistema maior do qual ele faz parte.

2.1
ContextNet Core

O *middleware* ContextNet Core visa disponibilizar serviços baseados em contexto. Ele é composto de (i) nós estacionários, os gateways para internet; de (ii) nós móveis, que são os Mobile Hubs; e por (iii) dispositivos periféricos, como sensores e atuadores que se conectam com os mobile hubs, chamados M-OBJs.

A versão mais recente do ContextNet sobre Kafka, desenvolvida por (Wanous 2021) permitiu um desenvolvimento de aplicações mais complexas e escaláveis em comparação com a versão anterior e é versão utilizada nos testes deste projeto. O Apache Kafka é uma plataforma open-source de streaming distribuído capaz de subscrever, publicar, armazenar e processar streams (fluxos de dados de aplicações ou sinais de controle) em tempo real (Red Hat 2022).

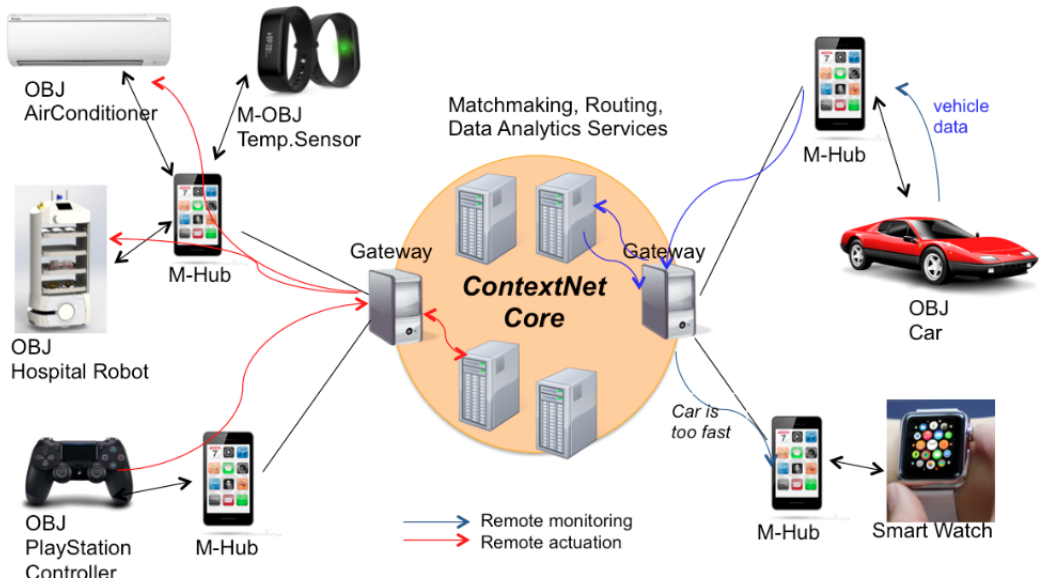


Figura 2.1: Arquitetura ContextNet Core (Talavera 2023)

2.2 Arquitetura do Mobile Hub

O Mobile Hub é organizado em uma arquitetura de multi-módulos, composto por módulos mandatórios e opcionais. Estes últimos podem ser incluídos de acordo com os requisitos da aplicação. Essa arquitetura torna mais fácil a implementação de novas tecnologias no código (Talavera 2023). Os módulos que compõem o M-Hub, como podem ser observados na figura 2.2 são a *core library*, *mobile hub service* e *mobile hub configurator*.

A biblioteca *Core* é um módulo mandatório na aplicação e lida com configurações do M-Hub e com as regras de negócio e interage com múltiplas interfaces quando recebe eventos (eg. novos dados de sensor) ou fazendo uma requisição a uma interface de uma biblioteca para que ela realize alguma ação. Cada biblioteca dessas suporta a múltiplas tecnologias de comunicação e funcionalidaes específicas.

- S2PA Gateway: interage com as tecnologias de curto alcance, gerenciando os dispositivos periféricos.
- Connection Gateway: interage com as tecnologias de longo alcance, enviando e recebendo informações da nuvem.
- Connection Gateway: realiza o mapeamento das mensagem para o controlador local.
- MEPA Gateway: processa os dados de várias fontes usando o Complex Event Processing (CEP), que é um conjunto de conceitos e técnicas para processamento de eventos em tempo real e extração de informação de streams de eventos a medida que eles cheguem.

Os *use cases* definem a lógica principal e os algoritmos do M-Hub, e interagem com as interfaces WLAN, que é implementada pelas tecnologias de comunicação de longa distância, WPAN, que lida com as comunicações de curta distância, e CEP, que gerencia a lógica de processamento.

O módulo *Core* deve ser configurado usando um objeto builder que define os plugins que devem ser incluídos na execução. Por exemplo, se for desejado utilizar apenas o BLE, sua biblioteca deve ser adicionada nas dependências e configurada com o objeto builder. Todas as dependências são gerenciadas pelo Mobile Hub Configurator, que também inicia um serviço no background que contém todos os gateways. Os gateways são responsáveis por manter o M-Hub executando em *background* e iniciar ou parar componentes.

2.2.1 S2PA Gateway

Esse gateway tem a responsabilidade de gerenciar múltiplos M-OBJs para receber/enviar informações de/para eles. Como o M-Hub deve interagir com múltiplas tecnologias WPAN ao mesmo tempo para abranger um maior número de dispositivos, o gateway mantém um único fluxo com todas as tecnologias, que começa com a constante descoberta de M-OBJs. O use case *scan* faz uma requisição para todos as tecnologias WPAN disponíveis para descobrir M-OBJs, onde cada tecnologia possui sua implementação específica do scan.

Assim que um M-OBJ é descoberto, o use case de conexão tenta iniciar uma conexão com o dispositivo. Se a conexão for bem sucedida, outros use cases poderão ser usados também, como o *subscribe*, que faz requisições ao M-OBJ para que ele forneça novos dados dos sensores, ou executar uma ação em um atuador.

2.2.2 Connection Gateway

O Connection Gateway é o ponto de entrada para qualquer comunicação e mapeia as mensagens para o controlador local. Um controlador é um componente que tem métodos CRUD (Create, Read, Update e Delete) para gerenciar operações relacionadas a regras de processamento, tipos de dados de sensor, mobile objects, etc.

Além disso, esse gateway também possui uma estratégia de buffering para as mensagens enviadas ao ContextNet Core.

2.2.3 MEPA Gateway

O CEP lida com todos os eventos de dados dos sensores, e gera eventos que são enviados para a nuvem com as tecnologias WLAN. O processamento local reduz a transmissão de eventos para a nuvem, só enviando dados que o sistema considera relevantes.

2.3 Protocolos WPAN

Atualmente o protocolo WPAN disponível no Mobile Hub é o BLE, e, como mencionado anteriormente, o objetivo deste trabalho é adicionar os protocolos Wi-Fi Direct e NFC, para a comunicação do Mobile Hub com os sensores e atuadores.

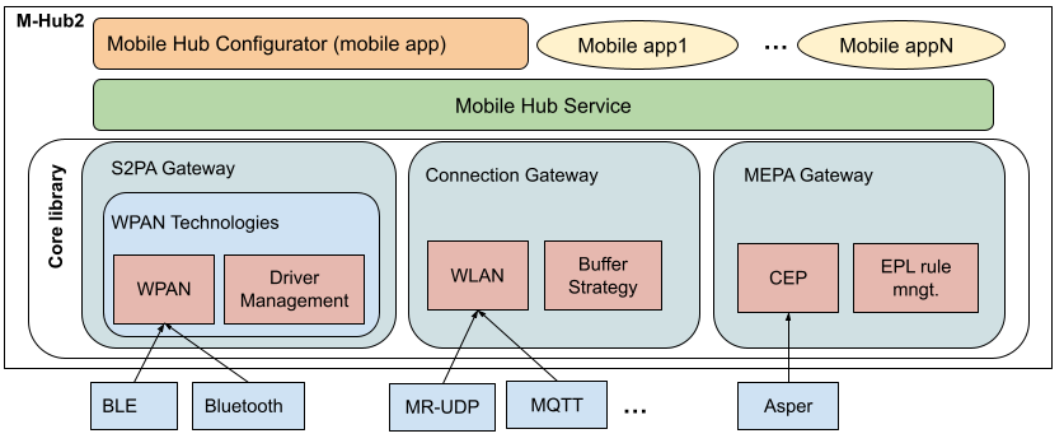


Figura 2.2: Arquitetura do Mobile Hub 2 (Talavera 2023)

No projeto, todos esses protocolos implementam a interface WPAN, que possui funções para descobrir dispositivos, conectar-se com eles, ler os dados continuamente, se conectar com os drivers, ler os dados uma vez e se desconectar do dispositivo.

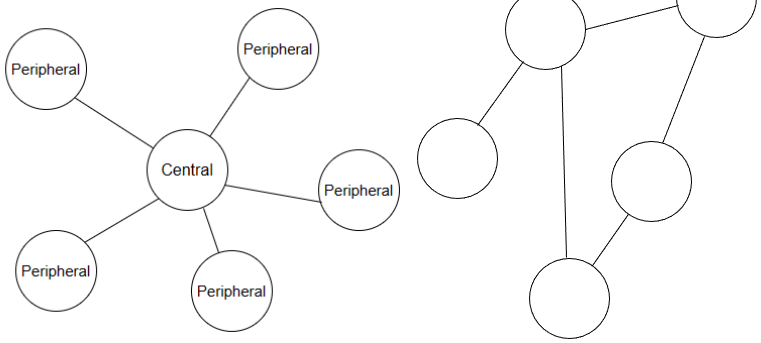
2.3.1 BLE

O BLE é uma variação do Bluetooth Clássico e foi desenvolvido para consumir menos bateria que o original. Para conseguir isso, o BLE usa uma frequência de rádio de baixa potência em comparação ao clássico e aumenta o intervalo de conexão, o que reduz a frequência de comunicação ativa. Os dispositivos BLE também passam a maior parte do tempo em estado ocioso ou de suspensão, e o protocolo emprega o salto de frequência adaptativo e o uso de protocolos eficientes de transferência de dados, como o Attribute Protocol (ATT) e o Generic Attribute Profile (GATT) (Araujo 2012).

Na fase de descoberta, um dispositivo envia um pacote de publicidade nos canais de escaneamento. Assim que outro dispositivo recebe o pacote, ele responde com um pacote de anúncio, que contém informações sobre os serviços e outras informações relevantes. Esses pacotes de anúncio são transmitidos em intervalos regulares nos canais de escaneamento.

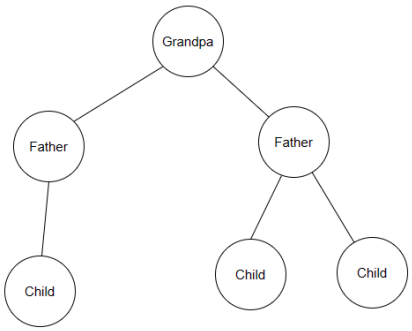
Quando um dispositivo tenta se conectar a outro, ele passa a ser um iniciador. Ao localizar um anunciador, ele envia uma solicitação de conexão ao mesmo, enviando uma solicitação de conexão. Quando a conexão é formada entre o iniciador e o anunciador, ambos passam a se comunicar através dos canais de dados.

No BLE existem três tipos de topologias possíveis (Labib 2019):



2.3(a): Estrela

2.3(b): Mesh



2.3(c): Árvore

Figura 2.3: Topologia das conexões BLE

– Estrela: existe um nó central e um ou mais nós periféricos. Cada nó periférico pode comunicar apenas com o nó central, não podendo se comunicar com outros periféricos diretamente.
– Mesh: cada dispositivo está conectado a um ou mais dispositivos.
– Estrutura de árvore: um dispositivo é o nó avô e os outros podem ser nós pais ou filhos. O nó avô, que é a raiz da árvore, pode ser comunicar com os nós pais, que por sua vez podem se comunicar com os nós filhos ligados a eles.

A topologia estrela tem a vantagem de ser mais simples e direta, além de possuir uma latência menor que as outras estruturas. Todavia, apresentea a desvantagem de possuir uma menor área de cobertura da rede devido ao limitado número de nós periféricos que podem ser conectados ao nó central.

Por sua vez, a topologia mesh é considerada a rede mais flexível e pode fornecer uma área maior de cobertura de rede com maior tolerancia a falhas. No entanto, ela consome mais energia que as outras redes e a latência é mais alta e imprevisível.

Já a estrutura de árvore permite que a rede inclua mais nós e tem a vantagem do roteamento ser mais simples que na topologia mesh. Comparada

à topologia estrela, a topologia de árvore consegue conectar mais nós e tem uma maior área de cobertura. As desvantagens dessa topologia são a alta taxa de latência quando existem muitos níveis na árvore, e a vunerabilidade da rede após um dos nós pai se desconectar (Li 2019).

O GATT define como ocorrerá a comunicação e transferência de dados no BLE, sendo os dispositivos divididos em cliente e servidor.

2.3.2 Wi-Fi Direct

O Wi-Fi Direct, inicialmente chamado de Wi-Fi Peer-to-Peer, foi contruído em cima da infraestrutura IEEE 802.11 (Wi-Fi), e oferece uma conexão direta, rápida e segura entre dispositivos, que podem ser *smartphones*, *laptops*, *smart TVs*, impressoras, câmeras e consoles de *video game* (eg. Xbox One).

Diferente do Wi-Fi, o Wi-Fi Direct não precisa de um ponto de acesso (Access Point) wireless ou de um roteador. Ele opera na frequência de 2.4 GHz, e tem alcance de até 200 m (Wi-Fi Alliance 2023) e tem velocidade de transferência de dados de até 250 Mbps (megabits por segundo) (Wi-Fi Alliance 2023).

Durante a fase de descoberta, um dispositivo envia uma requisição pedindo o endereço MAC dos dispositivos próximos. Com isso, outros dispositivos escutam a mensagem e respondem com uma mensagem unicast para o dispositivo que enviou.

A conexão por Wi-Fi Direct ocorre por estabelecimento de grupos, tal que um dispositivo é o Group Owner (GO) e os outros dispositivos são clientes (Camps-Mur 2013). Os clientes podem apenas se comunicar com o GO, de modo que a topologia do grupo é de estrela, como visto na figura 2.4. Essa topologia, que é muito próxima à do funcionamento do IEEE 802.11 (Wi-Fi), restringe a mobilidade.

Existem três técnicas de formação do grupo:

– Standard: os dispositivos devem descobrir um ao outro e negociar qual deles será o GO. Durante a fase de negociação, cada um dos dispositivos envia um Intent Value, que é um número entre 0 e 15, e o dispositivo que enviar o maior número se torna GO. A especificação do protocolo não define mecanismos para a seleção do Intent Value, deixando a cargo dos desenvolvedores essa tarefa (Khan 2016) [1];

[1]Possíveis formas de determinar o Intent Value estão relacionadas à capacidade do próprio dispositivo, como nível de bateria, capacidade de processamento, RSSI e distância do dispositivo em relação aos outros. Cabe ao desenvolvedor decidir como esses dados serão considerados na geração do Intent Value.

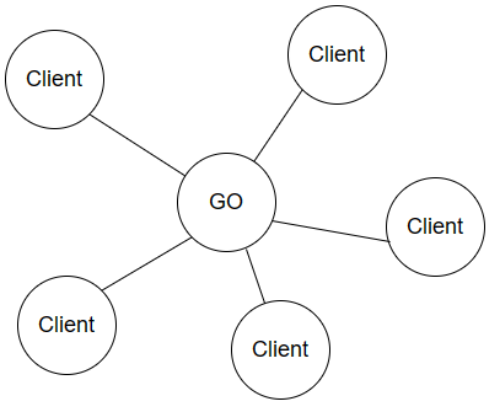


Figura 2.4: Topologia das conexões Wi-Fi Direct

– Autonomous: um dispositivo cria um grupo de forma autônoma e se torna o GO, dessa forma, outros dispositivos podem descobrir o grupo e se tornarem clientes do mesmo;
– Persistent: um dispositivo declara o grupo como persistente, assim não ocorrerá a fase de negociação e o grupo é restabelecido como anteriormente.

Durante a conexão não é possível mudar os cargos dos dispositivos, se for necessário que um cliente vire GO ou que um GO vire cliente, todas as conexões deverão ser desfeitas para que ocorra o processo de formação de grupo novamente.

É importante comentar que como o cargo de GO não pode ser transferido, se ele se desconectar do grupo, todas as conexões são quebradas e o grupo precisa ser formado novamente.

Assim que o grupo é formado, o GO continuamente anuncia sua presença enviando beacons. O GO deve então rodar o servidor Dynamic Host Configuration Protocol (DHCP) para alocar endereços IP para todos os clientes do grupo.

A tranferência de dados entre GO e cliente pode ocorrer por vários protocolos, como Transmission Control Protocol (TCP/IP) ou User Datagram Protocol (UDP).

2.3.3 NFC

NFC é um padrão de comunicação sem fio que permite que dois dispositivos troquem dados entre si estão fisicamente muito próximos um do outro, em uma distância de até 10 cm.

A comunicação é feita por radiofrequência, a partir da faixa de 13,56 MHz, com a taxa de transmissão de dados variando entre 106, 212 e 424 Kb/s (kilobits por segundo). Algumas aplicações podem trabalhar com taxa máxima de 848 Kb/s, mas esse não é o padrão oficial (Ylinen 2009).

No NFC, a conexão pode ocorrer entre dois smartphones ou entre um smartphone e uma tag NFC, onde a tag NFC pode ser um cartão, adesivo, smartwatches, billboard eletrônico, ou seja, qualquer objeto que tenha um chip NFC.

Existem cinco tipos de tags NFC:

- Tipo 1: normalmente, armazena entre 96 bytes e 2 KB de dados e tem taxa de transferência de 106 Kb/s;
- Tipo 2: armazena entre 48 bytes e 2 KB de dados e tem velocidade de 106 Kb/s. É compatível com o tipo 1;
- Tipo 3: baseada em uma tecnologia da Sony chamada FeliCa, esse tipo normalmente armazena 2 KB (mas pode chegar a 1 MB) e tem velocidade de 212 Kb/s. A compatibilidade com outros padrões existe, mas não é garantida;
- Tipo 4: armazena até 32 KB e tem taxa de transferência entre 106 Kb/s e 424 Kb/s. É compatível com os tipos 1 e 2;
- Tipo 5: é o tipo mais recente, apresentado em 2015. Usualmente, esse tipo de tag armazena até 64 KB e tem velocidade de 106 Kb/s.

O NFC tem dois modos de comunicação. No modo passsivo, um dispositivo atua como iniciador e o outro como um alvo. O dispositivo iniciador sempre fornece energia necessária para a comunicação e troca de dados entre os dois. No modo ativo, ambos os dispositivos possuem suas próprias fontes de energia e podem iniciar a comunicação.

Na fase de iniciação ocorrem a detecção, onde dispositivos NFC detectam outros dispositivos próximos por meio de uma antena, a inicialização de campo, em que os diapositivos ativam seus campos magnéticos para criar um campo de radiofrequência compartilhado, a negociação, em que os dispositivos inicial um processo de negociação para determinar o modo de operação a ser utilizado.

Existem três modos de operação:

- Leitura (Reader/Writer): um dispositivo NFC atua como leitor ou escritor e interage com uma tag ou cartão NFC passivo (é capaz de receber informações de outros dispositivos, mas não de iniciar a comunicação por conta própria) para ler ou gravar informações;

– Emulação de cartão (Card Emulation): o dispositivo NFC emula um cartão NFC, permitindo que seja usado como uma forma de pagamento ou autenticação. Ele pode ser reconhecido por um leitor NFC ativo;
– Peer-to-Peer (P2P): dois dispositivos NFC podem se comunicar entre si, trocando dados em ambas as direções. Isso permite a transferência de arquivos, compartilhamento de informações ou estabelecimento de uma conexão mais complexa.

A troca de dados entre dispositivos NFC e tags usa o NFC Data Exchange Format (NDEF). Toda mensagem NDEF contém um ou mais NDEF records. Cada record tem um header e um payload. O header possui tipo, ID do payload e tamanho, enquanto o payload contém os dados. NDEF suporta dados como: texto, URI, Smart Posters, Signatures, MIME (Igoe 2014).

2.4 Protocolos WLAN

Os protocolos WLAN são utilizados para a comunicação com o Gateway, e é possível escolher utilizar MR-UDP ou MQTT.

2.4.1 MR-UDP

O MR-UDP é um protocolo de transporte baseado em Reliable UDP de/para nós móveis, com o menor overhead possível. Ele permite a comunicação confiável com clientes móveis localizados atrás de Firewalls com NAT, e que podem trocar dinamicamente seus endereços IP ou porta de conexão (Nery e Silva 2013).

O MR-UDP foi desenvolvido para a manutenção de um grande número de conexão de nós móveis e a entrega confiável de mensagens de/para esses nós. Ele suporta aplicações onde todos os nós móveis enviam periodicamente dados de contexto em intervalos discretos em vez de streaming.

Com intuito de evitar a perda de pacotes que acontece em protocolos baseados em UDP, o MR-UDP define um tempo de espera curto e configurável entre envios consecutivos no lado do transmissor. Esse controle de fluxo é efetivo na maioria dos casos e não requer que nenhum estado de lógica de conexão seja mantido.

Funcionalidades como acknowledgment of received data packets (Ack), entrega de pacotes em ordem, retransmissão seletiva dos pacotes perdidos e over-buffering foram herdadas do R-UDP.

2.4.2 MQTT

O MQTT é um protocolo da camada de aplicação que tem as características de ser leve e eficiente, escalável, confiável, seguro e possui bom suporte de várias linguagens de programação. Foi desenvolvido para conexões em localização remota com dispositivos que possuem recursos limitados ou limitação de banda. Ele deve rodar em cima do protocolo de transporte TCP/IP.

O protocolo é baseado no modelo publish/subscribe, em que os componentes do sistema podem se comunicar de forma assíncrona e desacoplada. Os *publishers*, chamados de clientes MQTT, são responsáveis por enviar mensagens ou eventos para um canal de comunicação centralizado, conhecido como tópico. Já os *subscribers*, registram-se para receber mensagens de um ou mais tópicos. Dessa forma, quando um publisher envia uma mensagem para um tópico, todos os *subscribers* a recebem de forma assíncrona.

O MQTT ainda usa um terceiro componente, chamado *broker*, para filtrar todas as mensagens que chegam de *publishers* e distribuí-las corretamente para os *subscribers*.

Os tipos de mensagens são Connect, Disconnect e Publish. O *Connect* espera o estabelecimento de uma conexão com o *broker* e cria um link entre os nós. O *Disconnect* espera o cliente MQTT terminar o que ele está fazendo para então desconectar a sessão TCP/IP. Por fim, o *Publish* retorna imediatamente para a thread da aplicação após passar a requisição para o cliente MQTT.

As conexões MQTT acontecem sempre entre um cliente e um *broker*, o que significa que clientes nunca se conectam diretamente com outros clientes. Para iniciar uma conexão, o cliente envia uma mensagem CONNECT ao *broker*, que por sua vez responde com uma mensagem CONNACK e um código de status. Assim que a conexão é estabelecida, o *broker* matém ela aberta até que o cliente envie um comando de desconexão ou que a conexão se quebre.

Quality of Service (QoS) é o uso de mecanismos ou tecnologias que funcionam em uma rede para controlar o tráfego e garantir o desempenho de aplicativos essenciais com capacidade de rede limitada. O cliente que publica a mensagem para o *broker* define o nível de QoS da mensagem, quando ele envia a mensagem. O *broker* transmite essa mensagem, para os clientes inscritos usando o nível de QoS que cada cliente definiu. Se um desses clientes definir um QoS de nível mais baixo que o *publisher*, o broker transmitirá a menssagem no nível mais baixo de QoS. No MQTT existem três níveis de QoS: 0 (no máximo uma vez), 1 (pelo menos uma vez) e 2 (exatamente uma vez).

No nível 0 não existe garantia de entrega das mensagens e o receptor não envia um pacote de reconhecimento ao transmissor de que ele recebeu a

mensagem.

No nível 1 é garantido que a mensagem chegará pelo uma vez ao receptor, com isso, o transmissor que enviou guarda a mensagem até receber o pacote de confirmação PUBACK de que o receptor recebeu a mensagem. Se o transmissor não receber o pacote PUBACK no tempo definido, ele retransmite a mensagem para garantir sua entrega.

Já no nível 2, é garantido que cada mensagem é recebida somente uma vez pelo receptor. Essa garantia é fornecida por pelo menos dois request/response entre o transmissor e o receptor. Quando o receptor recebe um pacote PUBLISH de um transmissor, ele processa a mensagem e responde com um pacote PUBREC, que confirma que o receptor recebeu o pacote PUBLISH. Assim que o transmissor recebe o pacote PUBREC, ele discarta o pacote PUBLISH inicial. O transmissor então armazena o pacote PUBREC e responde o receptor com um pacote PUBREL. Assim, o receptor discarta todos os estados armazenados e responde com um pacote PUBCOMP. Se um pacote for perdido ao longo desse processo, o transmissor é responsável por retransmitir a mensagem (The HiveMQ Team 2015).

Isso é válido se o transmissor for cliente e o receptor for o *broker* e vice-versa.

As limitações do MQTT estão relacionadas à alta latência, alto uso de banda e consumo alto de energia.

3
Método Utilizado

Neste capítulo são apresentados os trabalhos relacionados, tanto com o Mobile Hub, quanto que os relacionados aos módulos desenvolvidos. Além disso, são mostradas como foram feitas os estudos de conceitos necessários para o entendimento, configurações iniciais do projeto e os testes iniciais com o BLE.

3.1
Trabalhos Relacionados

A utilização maciça de smartphones permite aplicações IoT potencialmente novas, generalizadas e participativas, em que os utilizadores de smartphones podem atuar como hubs universais para detecção ou atuação em M-OBJs mais simples (ou redes em malha dos mesmos) que apenas dispõem de conectividade sem fios de curto alcance, como ZigBee ou Bluetooth LE. Estes M-OBJs, que podem até ser móveis (ligados a veículos, animais de estimação ou outros bens), podem tirar partido da eventual proximidade da extremidade móvel (M-Hub) para a utilizar oportunamente como intermediário para transmitir dados e/ou receber parâmetros de comando de/para serviços IoT de retaguarda.

Por exemplo, uma extremidade móvel pode obter várias informações (temperatura, humidade, concentração de CO, etc.) de nós sensores em torno do utilizador, tanto para lhes disponibilizar essas informações como para as transmitir através da Internet móvel (3G/4G ou Wi-Fi) para um ambiente público.

Uma vez que esses hubs de borda não são estacionários, mas movem-se, a acessibilidade dos M-OBJs dispersos (sensores e atuadores) é intermitente, temporária, oportunista e automática (sem intervenção do utilizador). Devido à sua capacidade de se deslocar por diferentes regiões, é provável que um dispositivo de extremidade móvel, como o nosso M-Hub, encontre diferentes tipos de dispositivos/sensores IoT que apenas suportam uma tecnologia WPAN específica. Por outro lado, pode ter de interagir com serviços de backend que exigem ligações utilizando diferentes tecnologias WWAN ou WLAN.

Por conseguinte, o suporte de tecnologias sem fios heterogêneas é fun-

damental. Além disso, uma vez que estas tecnologias operam de formas ligeiramente diferentes, é importante dispor de um sistema de middleware de comunicações e processamento que possa ocultar estas diferenças através de interfaces de programação abrangentes para aceder e interagir com a pletora muito heterogênea de M-OBJ.

Complementar a variabilidade do contexto ambiental, os dispositivos móveis de ponta, como os smartphones, também enfrentam limitações de hardware ao mesmo tempo que fornecem uma computação eficaz. Para lidar com esses desafios, técnicas de descarregamento de computação são popularmente usadas (Rahmani 2021). Os algoritmos que as implementam melhoram a gestão de tarefas de descarga (Wang 2018), a atribuição de potência de transmissão (Yang 2018), a atribuição de recursos (Yuyi 2017) e a economia de energia (Zhang 2016) (Munoz 2015).

Apesar de todas as melhorias na computação em edge, a consciência da mobilidade ainda é um problema significativo, pois a maioria dos trabalhos existentes assume que os utilizadores estão estacionários durante a atribuição de tarefas e que a comunicação entre dispositivos móveis e nós de edge está sempre disponível (Maray 2022).

Pereira *et al* (Pereira 2013) defendem que os telefones celulares podem atuar como gateways de IoT, devido aos seus altos recursos atuais de computação e comunicação. Os dispositivos de sua arquitetura de rede são nós sensores BLE e telefones celulares padrão - atuando como pontos de acesso à Internet - que podem executar funções na borda que complementam o processamento na nuvem, como filtragem de dados, gerenciamento de alarmes e outros, para permitir o processamento distribuído.

Os autores também mencionam a importância do grande conjunto de sensores incorporados em telefones celulares para a IoT, pois eles permitem aplicações de automação de monitoramento e controle em uma ampla gama de domínios, como saúde e indústria 4.0. Eles observam que o uso de smartphones como gateways/hubs é necessário, pois várias das soluções atuais de IoT não conseguem lidar com cenários em que muitos (ou a maioria) dos objetos são móveis, como em VANETs, enxames de drones, rovers etc. Sua abordagem visa a uma arquitetura de rede holística para a troca e o processamento de dados de sensores e atuadores com a Internet, fazendo uso de serviços da Web RESTful incorporados e, ao mesmo tempo, oferecendo suporte à verdadeira mobilidade. Eles usam o CoAP (Constrained Application Protocol) e o formato de dados EXI (Efficient XML Interchange) para atingir seu objetivo.

Perera *et al* (Perera 2013) propõem uma aplicação móvel - Mobile Sensor Hub (MoSHub) - que permite que um telefone celular se conecte a

uma variedade de sensores diferentes e, portanto, colete, combine, processe e envie dados de sensores para um servidor. Uma arquitetura de software foi desenvolvida para interconectar dinamicamente sensores a um aplicativo móvel (em smartphones), gerando uma classe de invólucro chamada Sensor Device Definition (SDD), que descreve os recursos do sensor, e um sensor virtual, que oculta os detalhes de implementação e o acesso aos dados do sensor.

O trabalho de Zachariah *et al* (Zachariah 2015), previu que qualquer dispositivo BLE poderia aproveitar qualquer smartphone como um roteador de IP temporário e atuar como um host final de IP normal. Assim, eles propõem uma abordagem centrada no smartphone, em que qualquer telefone poderia fazer proxy de um perfil Bluetooth (serviços, características e atributos) para a nuvem em nome de um dispositivo. Eles também sugerem uma possível nova função para o smartphone, como um provedor de contexto oportunista para os dispositivos próximos que podem solicitar determinados serviços, como localização ou hora atual, e explicam que cada dispositivo de IoT deve fornecer algumas metainformações (conteúdo, tipo, destino etc.) em seu pacote de anúncios que determinam como o telefone deve fazer o proxy dos dados do perfil BLE. Em uma análise recente, Zachariah *et al* (Zachariah 2022) apesar do endosso de que a infraestrutura móvel ajuda a aliviar o problema do gateway, identifica os sistemas operacionais dos smartphones como um limite para a conectividade de longo prazo e tolerante a atrasos devido às otimizações de desempenho. Para lidar com essas restrições, os autores propuseram uma abordagem de gateway estático, implementado como um dispositivo autônomo usando o SoC ESP32 BLE/Wi-Fi.

Para facilitar o uso de smartphones como bordas móveis, vários trabalhos usam a plataforma Android, principalmente devido à sua natureza aberta e abrangente, que pode ser adaptada aos objetivos do aplicativo e ao hardware disponível, tornando todo o sistema de borda ainda mais eficiente. Zheng et al. (Zheng 2014) propõe o BraceForce, uma plataforma de middleware que incorpora conceitos orientados por eventos e modelos para fornecer abstrações de acesso eficientes e mais simples aos dispositivos de detecção em aplicativos móveis. Eles exigem que os dispositivos sensores implementem uma interface Java padrão (driver de sensor) para serem manipulados pelo middleware. Essa API contém comandos essenciais (por exemplo, abrir, fechar), configurações (por exemplo, iniciar, reiniciar) e primitivos de comunicação para enviar e receber dados de valor-chave. Entretanto, o BraceForce não aborda nem fornece operações para recuperar ou restaurar o estado de um sensor. Embora o BraceForce possa descobrir dispositivos sensores, ele só pode fazer isso para dispositivos conhecidos, ou seja, usando drivers de sensores pré-desenvolvidos.

Portanto, ao contrário da abordagem do Mobile Hub, ele não é capaz de se adaptar (carregar/descarregar módulos de sensores) para acessar dispositivos de sensores previamente desconhecidos.

3.2 Estudos Conceituais

A linguagem utilizada no desenvolvimento do Mobile Hub é Kotlin, que é uma linguagem de código aberto, estaticamente tipada que suporta programação orientada a objetos e programação funcional (Android Developers). Kotlin é compatível com Java, podendo utilizar todas as bibliotecas, com a vantagem de possuir sintaxe mais simples e menos verbosa (TechTarget 2022).

Foi necessário também estudar design patterns e conceitos de programação orientada a objetos, em especial Builder e Observer, dando continuidade aos padrões adotados anteriormente no projeto.

Além disso, o estudo teórico de todos os protocolos utilizados no Mobile Hub foi essencial para o desenvolvimento e entendimento das API's utilizadas.

3.3 Configuração do Projeto

No início foi feita toda a configuração do projeto, clonando o código-fonte do repositório e realizando a instalação das ferramentas acessórias.

Para a configuração do ContextNet Core foi necessária a instalação de uma imagem *Docker*, que contém os containers dos componentes utilizados para o ContextNet Core sobre Kafka, como mostrado na figura 3.1. Para o MQTT, é utilizado o broker Mosquitto [1], que é um software de código aberto que atua como um broker MQTT.

O aplicativo foi desenvolvido no Android Studio, um ambiente de desenvolvimento integrado (IDE) para o desenvolvimento de aplicativos Android.

O aplicativo Mobile Hub possui três telas: inicial, context data e configurações.

Na tela de configurações, figura 3.2(c), é possível escolher se a tecnologia WLAN utilizada será MQTT ou MR-UDP, sendo necessário também incluir no campo "Port" a porta do protocolo a ser utilizado (1883 para MQTT e 5500 para MR-UDP). Além disso, é preciso incluir no campo "IP Address" o IP da rede em que o computador que está rodando o ContextNet Core ou o mosquitto está conectado.

[1] https://mosquitto.org/

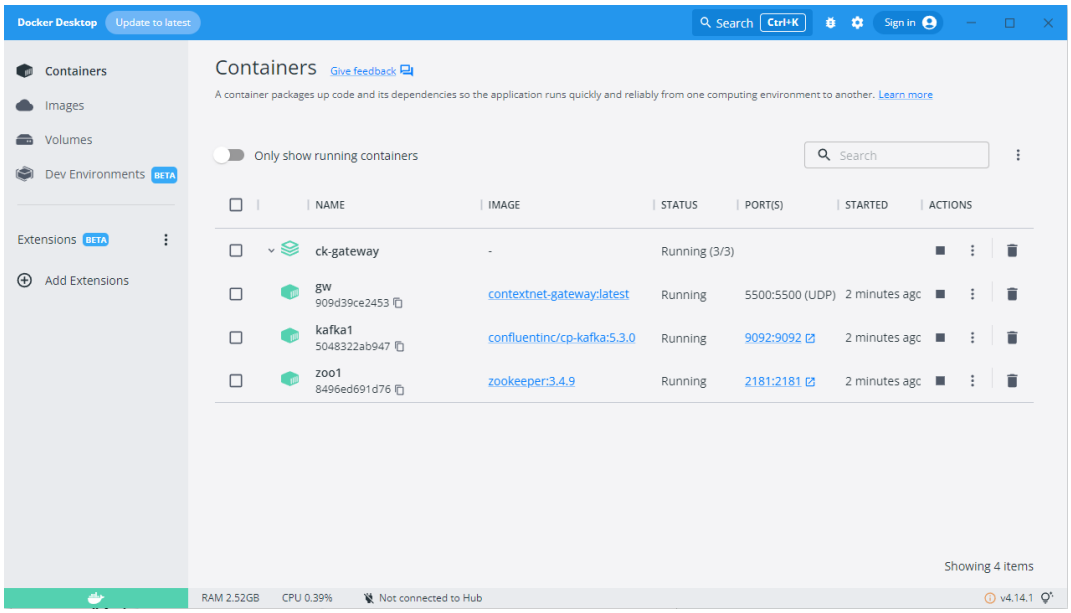


Figura 3.1: Containers Docker ContextNet Core

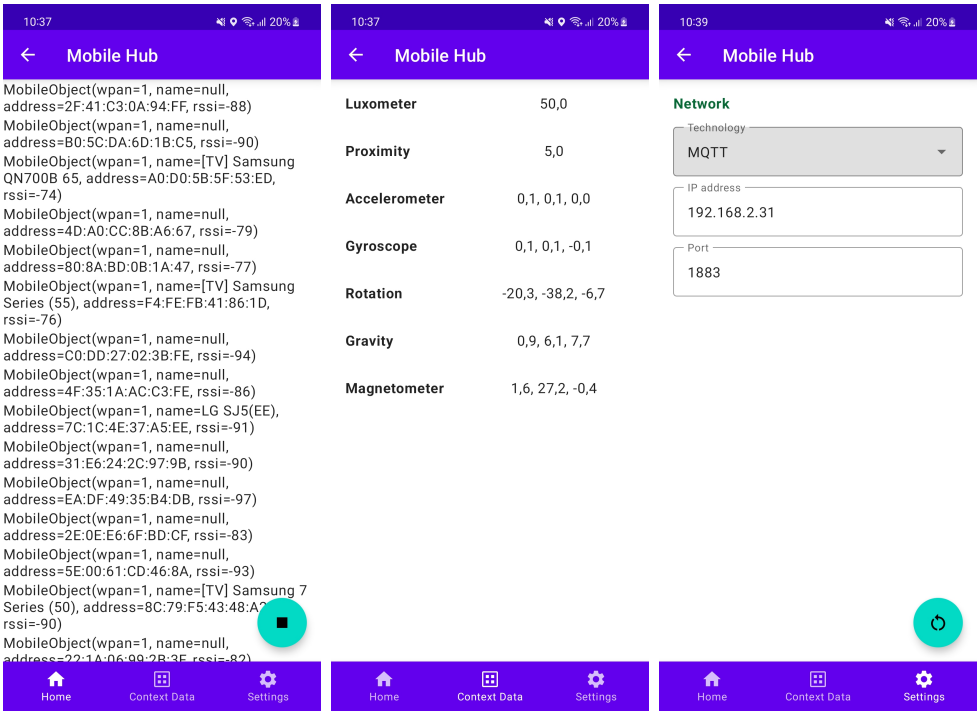
Na tela inicial, quando o Floating Button é acionado, o Mobile Hub chama os métodos do módulo BLE e os M-OBJs encontrados são "printados" na tela inicial do aplicativo, como mostrado na figura 3.2(a).

Na tela de Context Data, figura 3.2(b), são mostrados os dados dos sensores do smartphone que está rodando o aplicativo do Mobile Hub.

3.4 Testes Iniciais

Inicialmente foram realizados testes para entender a implementação do módulo BLE, já que a mesma lógica seria utilizada para os módulos a serem desenvolvidos. Posteriormente, ampliando para testes de tempo de conexão e de handover. Em ambos os testes foram utilizados, um SensorTag [2] (figura 3.3), o smartphone Samsung Galaxy A51 rodando Android 1L Snow Cone para o aplicativo do Mobile Hub e um computador com a configuração intel(R) Core (TM) i5-10400F CPU 2.90GHz with 4GB of RAM rodando Ubuntu 20.04.5 para os Gateways. Além disso, para os testes de handover, foi utilizado um segundo smartphone Samsung Galaxy A01 Core, rodando Android 10 Quince Tar, para o outro M-Hub. Esses dados foram utilizados também em (Talavera 2023).

[2]Texas Instruments CC2650 SensorTag - https://www.ti.com/lit/ml/swru410a/swru410a.pdf?ts=1678778276260



3.2(a): Tela inicial 3.2(b): Context Data 3.2(c): Configurações

Figura 3.2: Screenshots das telas do Mobile Hub

Figura 3.3: CC2650 SensorTag

3.4.1 Teste de Tempo de Conexão

Nos testes de conexão, foi avaliado tanto para o MQTT quanto para o MR-UDP, o tempo de conexão do Mobile Hub com o Gateway (CoT), o tempo em que o Gateway recebe o primeiro dado do sensor, e o tempo total, que é a soma dos dois tempos anteriores (TTR = CoT + TR). O experimento foi feito 10 vezes e foram calculados média e desvio padrão. Os resultados são mostrados na tabela 3.1, para o MQTT, e na tabela 3.2 para o MR-UDP.

Tabela 3.1: Tempo de conexão do M-Hub para o MQTT

MQTT	**CoT (s)**	**TR (s)**	**TTR**
Média	1.185	4.694	5.879
Desvio Padrão	0.09594	2.29174	

Tabela 3.2: Tempo de conexão do M-Hub para o MR-UDP

MR-UDP	**CoT (s)**	**TR (s)**	**TTR**
Média	0.592	4.17	4.762
Desvio Padrão	0.17623	0.60303	

Foi observado que o MR-UDP é duas vezes mais rápido que o MQTT em estabelecer a conexão com o ContextNet Core.

3.4.2 Teste de Handover

Nos testes de *handover*, foi avaliado tanto para o MQTT quanto para o MR-UDP, o tempo de receber o primeiro dado após cocorrer o handover (TR). O experimento foi feito 10 vezes e foram calculados média e desvio padrão. Os resultados são mostrados na tabela 3.3, para o MQTT, e na tabela 3.4 para o MR-UDP.

Tabela 3.3: Performance de handover para o MQTT

MQTT	**TR (s)**
Média	5.272
Desvio Padrão	0.42565

Tabela 3.4: Performance de handover para o MR-UDP

MR-UDP	**TR (s)**
Média	5.541
Desvio Padrão	0.27787

Os resultados indicam que há um aumento de TR devido ao handover. Já que ele é mais proeminente no MR-UDP, o valor de TR obtido com esse protocolo foi maior que o obtido com MQTT.

4
Projeto e Implementação

Neste capítulo é explicado como foram desenvolvidos os módulos Wi-Fi Direct e NFC.

4.1
Módulo Wi-Fi Direct

Para a implementação do módulo Wi-Fi Direct, foi utilizada a API `android.net.wifi.p2p` [1]. Essa API disponibiliza interfaces e métodos necessários para descobrir, conectar e trocar dados entre dispositivos usando o Wi-Fi Direct.

A classe principal do módulo Wi-Fi Direct, `WiFiDirectWPAN`, implementa os métodos da interface WPAN: `startScan`, `connect`, `connectWithDriver`, `subscribe`, `read` e `disconnect`.

Na inicialização da classe, no método `init`, são feitas as configurações iniciais da classe `WifiP2pManager`, que é responsável por controlar todas as operações relacionadas ao Wi-Fi Direct, como descoberta de dispositivos, conexão e transferência de dados, e do `WifiP2pManager.Channel`, que é usado para realizar transações e receber notificações sobre o status das operações. Em seguida, o grupo autônomo é formado. Foi escolhida essa técnica de formação de grupo, pois o M-Hub sempre precisará ter o papel de Group Owner. Dessa forma, criar o grupo autônomo economiza tempo de conexão, já que não haverá o processo de negociação entre o M-Hub e os M-OBJs.

No método `startScan`, os dispositivos periféricos são encontrados por meio de um `Broadcast Receiver`.

`Broadcast Receiver` é um dos principais componentes em aplicações Android. Ele tem a função de enviar ou receber mensagens de outras aplicações ou do próprio sistema, sendo essas mensagens events ou intents (Kantamani 2021).

Events são responsáveis pela ação de algum objeto conforme o usuário interage com o software, seja clicando, pressionando alguma tecla etc. Já

[1] https://developer.android.com/reference/kotlin/android/net/wifi/p2p/package-summary

intents são mensagens assíncronas que permitem que os componentes de um aplicativo solicitem a funcionalidade de outros componentes do Android.

Assim, no módulo, o `Broadcast Receiver` é usado para capturar eventos do Wi-Fi Direct. Dessa forma, quando um evento relacionado ao protocolo ocorrer, como a descoberta de dispositivos próximos ou a conexão estabelecida com outro dispositivos, o `Broadcast Receiver` será acionado e o método `onReceive` será chamado. Dentro desse método, são tratadas as informações recebidas e são executadas as informações necessárias em resposta a esses eventos.

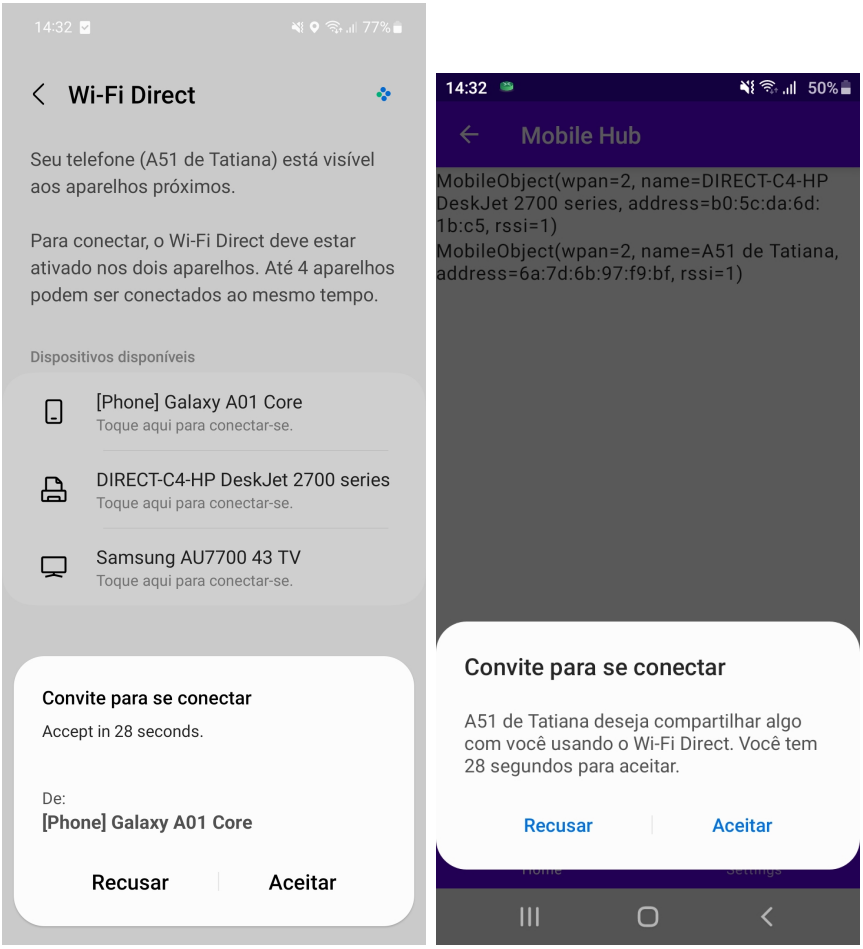
Lá, são criados os Mobile Objects, que são estruturas de dados pré-definidas utilizadas para identificar o dispositivo periférico descoberto. Elas possuem id (um número de identificação do protocolo WPAN utilizado para controle dentro do M-Hub), nome do dispositivo, endereço MAC e rssi (intensidade do sinal). Os Mobile Objects são então retornados pelo método `startScan`.

No método `connect`, é utilizado um método de conexão da classe `WifiP2pManager`, que usa o endereço MAC para enviar o convite para o dispositivo. Assim que o M-Hub pede a conexão do dispositivo periférico, aparece a tela de convite para se conectar, mostrado na figura 4.1(a). Se o M-OBJ aceitar a conexão em até 30 segundos, aparecerá na tela do dispositivo com Mobile Hub, uma outra tela de convite para se conectar com o dispositivo periférico, como na figura 4.1(b). Nesse método também é verificado se o dispositivo periférico entrou no grupo, e, se isso ocorrer, é possível iniciar a troca de dados. Os Mobile Objects são salvos em uma cache LRU, assim como foi feito no módulo BLE. Caso ocorra algum erro durante o processo de conexão, os M-OBJs são removidos da cache e poderão ser encontrados novamente pelo `startScan`.

A troca de mensagens entre GO e clientes foi feita por sockets no método `subscribe`. Os sockets são utilizados para estabelecer uma conexão bidirecional entre um cliente e um servidor, permitindo a troca de dados entre eles. Ele é identificado por um endereço IP e um número de porta, que combinados identificam exclusivamente um processo em uma rede (Oracle Java Documentation). Utilizou-se o socket TCP/IP pois este fornece uma conexão confiável, orientada a conexão e garante a entrega ordenada das mensagens.

A implementação foi feita utilizando a interface Java de Sockets, que disponibiliza as classes `Socket`, `ServerSocket`, `InputStream` e `OutputStream`.

É criado um `ServerSocket`, que espera por uma conexão de um cliente em uma porta específica com o método `accept`. Assim que o cliente aceita a conexão, o servidor passa a receber as mensagens transmitidas por ele. Para



4.1(a): M-OBJ 4.1(b): M-Hub

Figura 4.1: Pedido de Conexão Wi-Fi Direct entre M-Hub e M-OBJ

ler a mensagem recebida, é utilizado o método `ObjectInputStream`.

Por fim, se um M-OBJ se desconectar do grupo, isso é identificado por outro `Broadcast Receiver` no método `disconnect`.

O processo na classe `WiFiDirectWPAN`, desde a inicialização até a troca de dados, foi ilustrado no fluxograma da figura 4.2.

Existem limitações em relação à API. Quando a conexão é estabelecida, o Group Owner é sempre atribuído ao IP 192.168.49.1 (por DHCP) e os clientes são atribuídos o IP 192.168.49.x, onde x é um número aleatório entre 2 e 254. Dessa forma, um GO não pode se comunicar com outro GO, não possibilitando assim que um M-Hub se comunique com outro por Wi-Fi Direct no Android. Existem maneiras de contornar isso, por exemplo, tornando o M-Hub um cliente legado de outro grupo, como em (Casetti 2015), ou utilizando outros protocolos para esse tipo de comunicação, como visto em (Lee 2017).

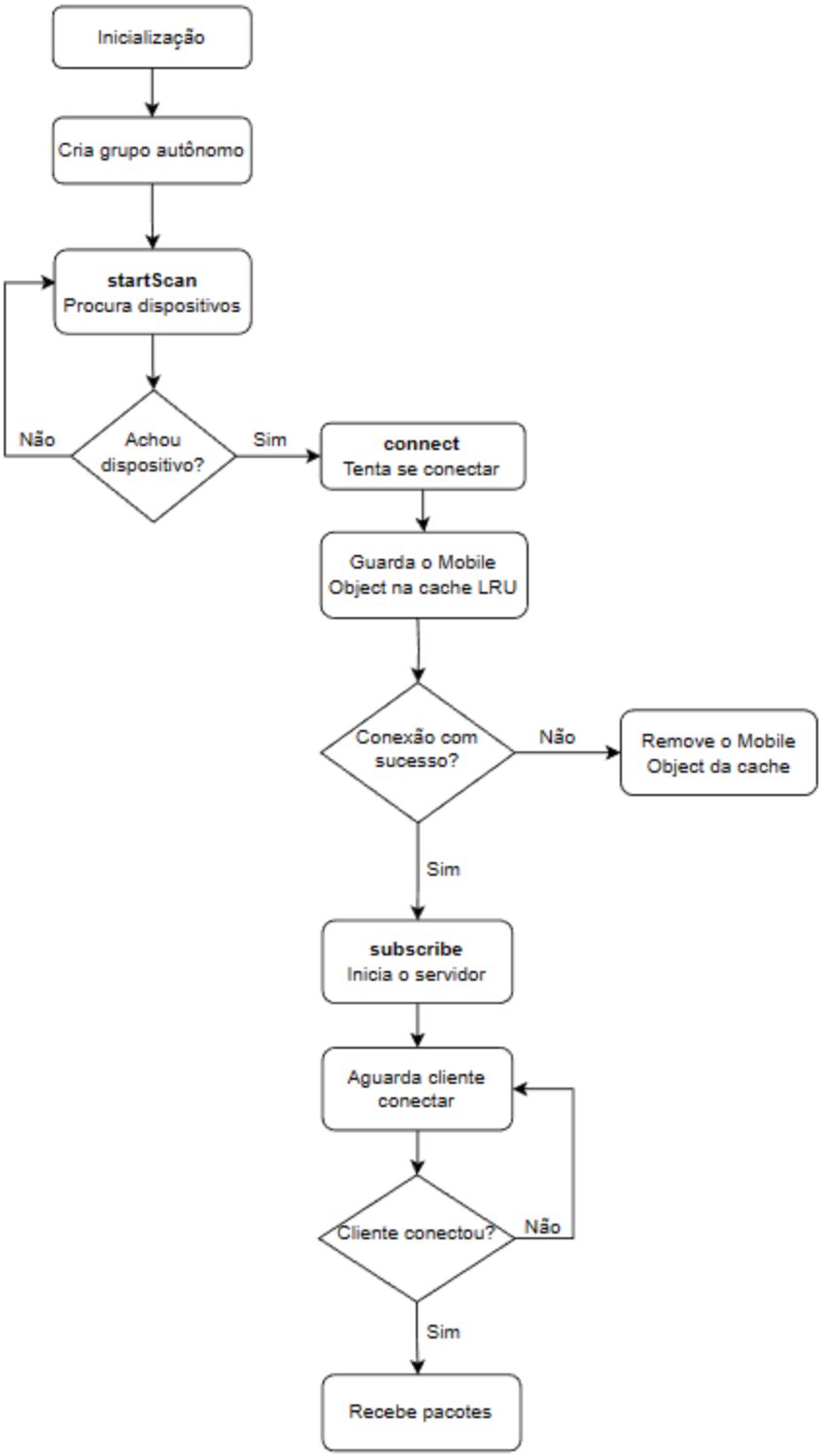


Figura 4.2: Fluxograma Wi-Fi Direct

4.2 Módulo NFC

Para a implementação do módulo NFC, foram utilizadas as APIs `android.nfc` [2], que fornece um conjunto de classes e interfaces que permite a interação com o NFC em dispositivos Android possibilitando a leitura de mensagens NDEF, e `android.nfc.tech` [3], que possui classes que fornecem acesso às características técnicas de uma tag.

Para a classe `NfcWPAN`, além de implementar a interface `WPAN`, também é preciso estender o componente `Activity`, pois o NFC necessita da interação onde o usuário está envolvido, como na aproximação de dois dispositivos ou do dispositivo com um cartão.

O componente `Activity` é um dos principais componentes do desenvolvimento Android. Ele representa uma única tela com uma interface de usuário, na qual os usuários podem interagir e realizar ações. Essa tela pode conter elementos de interface do usuário, como botões, campos de texto, imagens e listas.

Ele possui um ciclo de vida, que é um conjunto de estados pelos quais um componente passa desde a sua criação até a sua destruição. A medida que a `Activity` muda de estado, as callbacks são invocadas. São elas: `onCreate`, `onStart`, `onResume`, `onPause`, `onStop` e `onDestroy` [4]. Não é obrigatório passar por todos os estados e serão descritas aqui apenas as callbacks que foram utilizadas na classe `NfcWPAN`.

Os componentes `Activity` são chamados pelo sistema quando solicitados por meio de um intent ou quando um evento ocorre, e por isso, devem ser declarados no arquivo manifest da aplicação, que é um arquivo XML que fornece informações sobre o aplicativo ao sistema operacional Android, como componentes, permissões, recursos, configurações e declarações de metadados.

Quando um intent é usado, o sistema Android deve procurar o componente apropriado para que ele seja iniciado, comparando o conteúdo do intent aos intent filters declarados no arquivo manifest da aplicação. Se o intent encontrar o intent filter, o sistema inicia o componente encontrado e entrega o objeto intent. Os intent filters especificam os tipos de intents que o componente pode receber.

No caso da API NFC são usados os seguintes intent filters:

– `ACTION_NDEF_DISCOVERED` é filtrado quando uma tag que contém um NDEF payload é escaneada e tem tipo reconhecido;

[2] https://developer.android.com/reference/kotlin/android/nfc/package-summary
[3] https://developer.android.com/reference/kotlin/android/nfc/tech/package-summary
[4] Existem outras callbacks relacionadas a eventos específicos, mas essas são as principais.

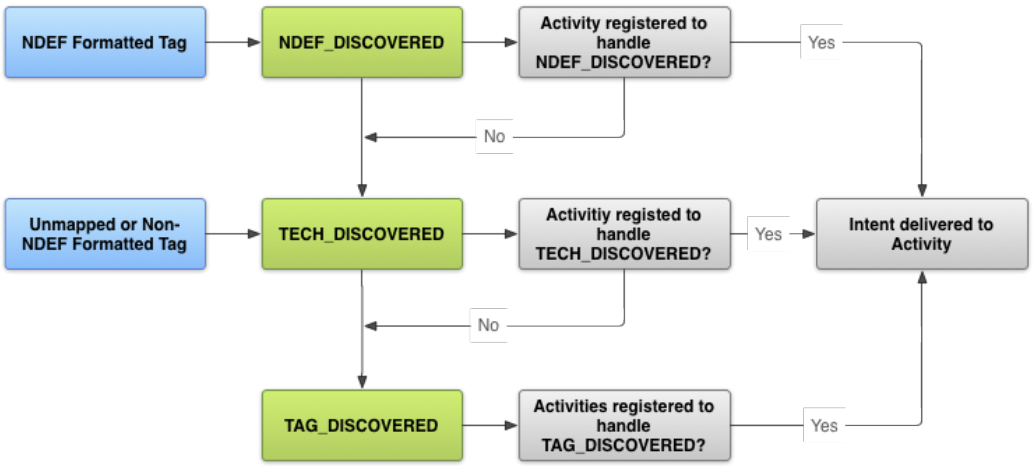


Figura 4.3: Descoberta de tags (Android Developers)

- `ACTION_TECH_DISCOVERED` é filtrado quando quando uma tag NFC que suporta tecnologias específicas (como MIFARE Classic, ISO-DEP, etc.) é descoberta.
- `ACTION_TAG_DISCOVERED` é filtrado quando uma nova tag NFC é descoberta. É o intent mais genérico pois pode descobrir qualquer tipo de tag;

A callback `onCreate` é invocada quando a `Activity` entra no estado Created, o estado inicial. Nesse método, deve ser feita a lógica que acontece uma única vez no ciclo de vida. Com isso, a classe `NfcAdapter` da API NFC é inicializada. Ela é o ponto de entrada para a interação com o hardware NFC do dispositivo, fornecendo métodos para verificar se o NFC está habilitado no aparelho e para registrar e desregistrar atividades como leitor ou gravador de tags NFC.

Quando a `Activity` entra no estado Resumed, ela aparece em primeiro plano, e o sistema invoca a callback `onResume`. Esse é o estado em que a `Activity` interage com o usuário. Aqui o método `enableForegroundDispatch` da classe `NfcAdapter` é chamado. Ele dá prioridade à essa `Activity` em relação a todas as outras atividades do sistema.

O método `onNewIntent` também faz parte do conjunto de callbacks do componente `Activity` e é chamado sempre que ocorre uma interação do dispositivo com um intent. Deve-se filtrar pelos intents na ordem de prioridade: (i) `ACTION_NDEF_DISCOVERED`, (ii) `ACTION_TECH_DISCOVERED` e (iii) `ACTION_TAG_DISCOVERED`. Um esquema disso é mostrado na figura 4.3.

Se o intent for `ACTION_NDEF_DISCOVERED`, os dados NDEF são lidos e armazenados no objeto `MobileObjectData` para serem utilizados no método `subscribe`.

Para todos os intents é necessário identificar qual tipo de tecnologia a tag está utilizando, que são NfcA (ISO 14443-3A), NfcB (ISO 14443-3B), NfcF (JIS 6319-4), NfcV (ISO 15693), IsoDep (ISO 14443-4), Ndef, NdefFormatable, NfcBarcode, MifareClassic e MifareUltralight. Cada tag possui uma classe e nela existem métodos para extrair suas propriedades. Essas informações são salvas no objeto Mobile Object.

No método `startScan` são simplemente retornados o Mobile Object concontrado.

No método `subscribe` são retornados os dados obtidos das tags que haviam sido armazenados no objeto `MobileObjectData`.

Ao contrário do módulo Wi-Fi Direct, não foram utilizados os métodos `connect` e `disconnect`, já que o M-Hub não permanece conectado à tag NFC. Assim, esses métodos apenas retornam.

5
Avaliação e Testes

Neste capítulo são mostrados os testes utilizando cada um dos protocolos implementados no projeto.

5.1
Testes Wi-Fi Direct

Para a realização dos testes Wi-Fi Direct foi desenvolvido um segundo aplicativo, também em Kotlin, para o M-OBJ a ser descoberto. Ele roda em background e coleta dados do sensor do acelerômetro do smartphone, enviando-os continuamente para o socket do Mobile Hub utilizando o IP 192.168.49.1 e porta 8988. Assim que ocorrer a conexão, o M-Hub passa a receber esses dados pelo socket.

No teste de conexão verificou-se que leva em média 5.89 segundos para o M-OBJ ser descoberto, com desvio padrão de 0.6223. O resultado foi medido 10 vezes e os dispositivos usados foram os mesmos que os dos testes iniciais.

Os testes Wi-Fi Direct com o M-OBJ sendo um dispositivo Android impossibilitam a realização de testes mais complexos que envolvam BLE e/ou NFC. Isso se deve pelo fato de que para um dispositivo Android ser encontrado por outro dispositivo utilizando Wi-Fi Direct, esse protocolo deve estar ativo. Isso só é possível enquanto o smartphone permanecer na tela do Wi-Fi Direct, que geralmente está localizada nas configurações avançadas da tela de Wi-Fi.

5.2
Testes NFC

Não foi possível testar o recebimento de mensagens NDEF pois seria necessário desenvolver um aplicativo para o M-OBJ que permitisse a conexão peer-to-peer. Foram testados aplicativos disponíveis na playstore, mas nenhum oferecia esse modo de operação.

6
Conclusão

Neste projeto foram implementados os protocolos WPAN Wi-Fi Direct e NFC no middleware Mobile Hub, utilizando as APIs Android, permitindo que o M-Hub se comunique com um maior número de dispositivos.

Futuramente, uma funcionalidade importante a ser desenvolvida é a comunicação entre M-Hubs. Se um M-Hub tiver sua comunicação com o ContextNet Core interrompida, ele poderia se comunicar com um outro M-Hub que está por perto, avisando o que ocorreu. Essa conexão poderia ser feita por BLE ou por Wi-Fi Direct (utilizando uma das técnicas mencionadas anteriormente).

Na parte das tecnologias WLAN, seria interessante que as escolha entre os protocolos fosse decidida pelo próprio M-Hub. Assim, a tecnologia seria escolhida de acordo com as características da rede e dos tipos de dados a serem transmitidos.

Por fim, a inclusão de mais protocolos WPAN e WLAN também seria importante para aumentar ainda mais o número de dispositivo com que o M-Hub pode se comunicar.

Referências Bibliográficas

[Android Developers] **Nfc basics**. [Online; accessed 21-June-2023].

[Android Developers] **Kotlin overview**. [Online; accessed 02-June-2023].

[Araujo 2012]

[Camps-Mur 2013] CAMPS-MUR, D.; GARCIA-SAAVEDRA, A. ; SERRANO, P.. **Device-to-device communications with wi-fi direct: overview and experimentation**. IEEE wireless communications, 20(3):96–104, 2013.

[Casetti 2015] CASETTI, C.; CHIASSERINI, C. F.; PELLE, L. C.; VALLE, C. D.; DUAN, Y. ; GIACCONE, P.. **Content-centric routing in wi-fi direct multi-group networks**. In: 2015 IEEE 16TH INTERNATIONAL SYMPOSIUM ON A WORLD OF WIRELESS, MOBILE AND MULTIMEDIA NETWORKS (WOWMOM), p. 1–9, 2015.

[IBM 2016] CLARK, JEN. **What is the internet of things (iot)?**, 2016. [Online; accessed 13-June-2023].

[Igoe 2014] IGOE, T.; COLEMAN, D. ; JEPSON, B.. **Beginning NFC: near field communication with Arduino, Android, and Phonegap**. "O'Reilly Media, Inc.", 2014.

[Kantamani 2021] SATYA PAVAN KANTAMANI. **Broadcastreceiver in android**. [Online; accessed 01-June-2023].

[Khan 2016] KHAN, M. A.; CHERIF, W. ; FILALI, F.. **Group owner election in wi-fi direct**. In: 2016 IEEE 7TH ANNUAL UBIQUITOUS COMPUTING, ELECTRONICS MOBILE COMMUNICATION CONFERENCE (UEMCON), p. 1–9, 2016.

[Labib 2019] LABIB, M.; GHALWASH, A.; ABDULKADER, S. ; ELGAZZAR, M.. **Networking solutions for connecting bluetooth low energy devices-a comparison**. In: MATEC WEB OF CONFERENCES, volumen 292, p. 02003. EDP Sciences, 2019.

[Lee 2017] LEE, J. H.; PARK, M.-S. ; SHAH, S. C.. **Wi-fi direct based mobile ad hoc network**. In: 2017 2ND INTERNATIONAL CONFERENCE ON

COMPUTER AND COMMUNICATION SYSTEMS (ICCCS), p. 116–120, 2017.

[Li 2019] STANFORD LI, YAN ZHANG, AND MARIE HERNES. **Bluetooth® low energy tree structure network**. [Online; accessed 19-June-2023].

[Maray 2022] MARAY, M.; SHUJA, J.. **Computation offloading in mobile cloud computing and mobile edge computing: Survey, taxonomy, and open issues**. Mobile Information Systems, 2022:17, 2022.

[Munoz 2015] O. MUNOZ, A. P.-I.; VIDAL, J.. **Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading**. IEEE Transactions on Vehicular Technology, 64(10):4738—–4755, 2015.

[Nery e Silva 2013] NERY E SILVA, L.; ENDLER, M. ; RORIZ, M.. **Mr-udp: Yet another reliable user datagram protocol, now for mobile nodes**. Technical Report 06, Departmento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Abril 2013.

[Oracle Java Documentation] **What is a socket?** [Online; accessed 02-June-2023].

[Pereira 2013] PEREIRA, P. P.; ELIASSON, J.; KYUSAKOV, R.; DELSING, J.; RAAYATINEZHAD, A. ; JOHANSSON, M.. **Enabling cloud connectivity for mobile internet of things applications**. In: 2013 IEEE SEVENTH INTERNATIONAL SYMPOSIUM ON SERVICE-ORIENTED SYSTEM ENGINEERING, p. 518–526. IEEE, 2013.

[Perera 2013] C. PERERA, P. JAYARAMAN, A. Z. P. C.; GEORGAKOPOULOS, D.. **"dynamic configuration of sensors using mobile sensor hub in internet of things paradigm**. IEEE Eighth International Conference on Intelligent Sensors, Sensor Networks and Information Processing, p. 473–478, 2013.

[Rahmani 2021] RAHMANI, A. M.; MOHAMMADI, M.; MOHAMMED, A. H.; KARIM, S. H. T.; MAJEED, M. K.; MASDARI, M. ; HOSSEINZADEH, M.. **Towards data and computation offloading in mobile cloud computing: taxonomy, overview, and future directions**. Wireless Personal Communications, 119(1):147–185, 2021.

[Red Hat 2022] **O que é apache kafka?** [Online; accessed 21-June-2023].

[Talavera 2015] TALAVERA, L. E.; ENDLER, M.; VASCONCELOS, I.; VASCONCELOS, R.; CUNHA, M. ; DA SILVA E SILVA, F. J.. **The Mobile Hub concept: Enabling applications for the Internet of Mobile Things**. In: PERCOM WORKSHOPS, p. 123–128, Mar. 2015.

[Talavera 2023] TALAVERA, L. E.; REIMER, T.; CAVALCANTI, M.; CANTERGIANI, G.; TEIXEIRA, M. A. ; ENDLER, M.. **Design and implementation of a flexible architecture for mobile edge devices**. Technical Report 04, Departmento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, May 2023.

[TechTarget 2022] LUTKEVICH, BEN. **Kotlin overview**. [Online; accessed 02-June-2023].

[The HiveMQ Team 2015] THE HIVEMQ TEAM. **Mqtt quality of service (qos) 0,1, 2 – mqtt essentials: Part 6**. [Online; accessed 16-June-2023].

[Wang 2018] Z. WANG, Z. ZHAO, G. M. X. H. Q. N.; WANG, R.. **User mobility aware task assignment for mobile edge computing**. Future Generation Computer Systems, 85:1–8, 2018.

[Wanous 2021] WANOUS, C. A.. **Reengenharia do ContextNet utilizando Kafka**. PhD thesis, Pontifícia Universidade Católica do Rio de Janeiro, 2021.

[Wi-Fi Alliance 2023] **How far does a wi-fi direct connection travel?** [Online; accessed 05-May-2023].

[Wi-Fi Alliance 2023] **How fast is wi-fi direct?** [Online; accessed 05-May-2023].

[Yang 2018] L. YANG, H. ZHANG, M. L. J. G.; JI, H.. **Mobile edge computing empowered energy efficient task offloading in 5g**. IEEE Transactions on Vehicular Technology, 67(7):6398—-6409, 2018.

[Ylinen 2009] YLINEN, J.; KOSKELA, M.; ISO-ANTTILA, L. ; LOULA, P.. **Near field communication network services**. In: 2009 THIRD INTERNATIONAL CONFERENCE ON DIGITAL SOCIETY, p. 89–93, 2009.

[Yuyi 2017] M. YUYI, Z. J.; LETAIEF, K. B.. **Joint task offloading scheduling and transmit power allocation for mobile-edge computing systems**. In: PROCEEDINGS OF THE WIRELESS COMMUNICATIONS AND NETWORKING CONFERENCE (WCNC), p. 1–6. IEEE, March 2017.

[Zachariah 2015] ZACHARIAH, T.; KLUGMAN, N.; CAMPBELL, B.; ADKINS, J.; JACKSON, N. ; DUTTA, P.. **The internet of things has a gateway problem**. In: PROCEEDINGS OF THE 16TH INTERNATIONAL WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, HotMobile '15, p. 27–32, New York, NY, USA, 2015. Association for Computing Machinery.

[Zachariah 2022] ZACHARIAH, T.; JACKSON, N. ; DUTTA, P.. **The internet of things still has a gateway problem**. In: PROCEEDINGS OF THE 23RD ANNUAL INTERNATIONAL WORKSHOP ON MOBILE COMPUTING SYSTEMS AND APPLICATIONS, HotMobile '22, p. 109–115, New York, NY, USA, 2022. Association for Computing Machinery.

[Zhang 2016] K. ZHANG, Y. MAO, S. L. E. A.. **Energy-efficient offloading for mobile edge computing in 5g heterogeneous networks**. IEEE Access, 4:5896—-5907, 2016.

[Zheng 2014] X. ZHENG, D. E. P.; JULIEN, C.. **"braceforce: A middleware to enable sensing integration in mobile applications for novice programmers**. In: PROCEEDINGS OF THE 1ST INTERNATIONAL CONFERENCE ON MOBILE SOFTWARE ENGINEERING AND SYSTEMS, p. 8–17, 2014.