UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS RUSSAS

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

MARCELO MOREIRA

ADAPTAÇÃO DA API NEARBY CONNECTIONS PARA USO DOS PADRÕES DE COMUNICAÇÃO REQUEST-REPLY, PUBLISH-SUBSCRIBE E PIPELINE

RUSSAS

2020

MARCELO MOREIRA

ADAPTAÇÃO DA API NEARBY CONNECTIONS PARA USO DOS PADRÕES DE COMUNICAÇÃO REQUEST-REPLY, PUBLISH-SUBSCRIBE E PIPELINE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Ms. Filipe Maciel Roberto

RUSSAS

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

M838a Moreira, Marcelo.
Adaptação da Api Nearby Connections para uso dos padrões de comunicação request-reply, publish-subscribe e pipeline / Marcelo Moreira. – 2020.
46 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Russas, Curso de Engenharia de Software, Russas, 2020.
Orientação: Prof. Me. Filipe Marciel Roberto.

1. Padrões de comunicação. 2. Nearby Connections. 3. Conexão P2P. 4. Rede Ad-Hoc. I. Título.
CDD 005.1

MARCELO MOREIRA

ADAPTAÇÃO DA API NEARBY CONNECTIONS PARA USO DOS PADRÕES DE COMUNICAÇÃO REQUEST-REPLY, PUBLISH-SUBSCRIBE E PIPELINE

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Russas da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em:

BANCA EXAMINADORA

Prof. Ms. Filipe Maciel Roberto (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Alexandre Matos Arruda
Universidade Federal do Ceará (UFC)

Prof. Dr. Dmontier Pinheiro Aragão Jr.
Universidade Federal do Ceará (UFC)

RESUMO

Em diversos cenários o uso da infraestrutura da internet pode não ser o melhor meio de troca de dados entre aplicações. Nesses casos, os dispositivos podem estar demasiadamente próximos a ponto de poderem trocar dados diretamente um com o outro. Porém, para que isso possa acontecer, exige-se que diversos desafios e problemas sejam solucionados. Estes problemas são foco de diversos tipos de redes e tecnologias, cada uma focando em solucionar parte desses problemas, como é o caso da API Nearby Connections. Esta API provê serviços de conexão e troca de dados off-line entre dispositivos. Porém, seus serviços são providos de modo bastante abstratos e pouco semânticos. Diante disso, este trabalho propõe uma melhoria para que três padrões de comunicação sejam disponibilizados pela API em questão. Desse modo, aplicações consumidoras do serviço da Nearby Connections poderão escolher papéis definidos pelos padrões introduzidos, tornando a API mais semântica.

Palavras-chave: Padrões de comunicação. Nearby Connections. Conexão P2P. Rede Ad-Hoc

ABSTRACT

In several scenarios the use of the internet infrastructure may not be the best means of exchanging data between applications. In such cases, the devices may be too close at the point that they can exchange data directly with each other. However, for this to happen, it is required that several challenges and problems are solved. These problems are the focus of several types of networks and technologies, each focusing on solving part of these problems, such as the Nearby Connections API. This API provides offline connection and data exchange services between devices. However, its services are provided in a very abstract and non-semantic way. Therefore, this work proposes an improvement so that three communication patterns are made available by the API in question. In this way, applications consuming the Nearby Connections service will be able to choose roles defined by the patterns introduced, making the API more semantic.

Keywords: Communication patterns. Nearby Connections. Connection P2P. Ad-Hoc Networks.

LISTA DE FIGURAS

Figura 1 – Cronograma do trabalho. 15
Figura 2 – Diagrama de uma rede convencional. 16
Figura 3 – Diagrama de uma rede Ad-Hoc. 17
Figura 4 – Exemplo de *MANET*. 17
Figura 5 – Diagrama do padrão request-reply. 19
Figura 6 – Diagrama do padrão publish-subscribe. 20
Figura 7 – Diagrama do padrão pipeline. 21
Figura 8 – Diagrama arquitetural da *API* proposta na *JSR 259*. 23
Figura 9 – Fluxo da Nearby Connections. 25
Figura 10 – Arquitetura proposta. 26
Figura 11 – Diagrama de componentes da proposta. 26
Figura 12 – Diagrama de classes da proposta. 28
Figura 13 – Divisões no diagrama de classes da proposta. 28
Figura 14 – Primeira parte da divisão no diagrama de classes da proposta. 29
Figura 15 – Segunda parte da divisão no diagrama de classes da proposta. 30
Figura 16 – Modificação no build.gradle do projeto. 30
Figura 17 – Modificação no build.gradle(Module: app). 31
Figura 18 – Modificações do arquivo Manifest.xml. 31
Figura 19 – Aplicação de testes. 33
Figura 20 – Componentes da aplicação de testes. 34
Figura 21 – Resultados do teste 1, comportamentos iguais. 36
Figura 22 – Resultados do teste 2A, Cruzamento de comportamentos. 36
Figura 23 – Resultados do teste 2B, Cruzamento de comportamentos. 37
Figura 24 – Resultados do teste 3, Tipos incompatíveis no mesmo padrão. 38
Figura 25 – Resultados do teste 4, Tipos compatíveis. 38
Figura 26 – Resultados do teste 5, Envio e recebimento de dados. 38
Figura 27 – Resultados do teste 6, Subscrição de serviço. 39
Figura 28 – Resultados do teste 7, Conclusão de processamento. 39
Figura 29 – Uso de memória RAM antes do inicio das aplicações de teste. 40
Figura 30 – Uso do processador antes do inicio das aplicações de teste. 41
Figura 31 – Uso de memória RAM durante o uso das aplicações de teste. 42

Figura 32 – Uso do processador durante o uso das aplicações de teste. 43
Figura 33 – Agilidade de desenvolvimento da aplicação A. 44
Figura 34 – Agilidade de desenvolvimento da aplicação B. 44

LISTA DE ABREVIATURAS E SIGLAS

AHNA-EG	*Ad Hoc Networking API Expert Group* / Grupo de Especialistas em API para Redes Ad-hoc
API	*Application Programming Interface* / Interface de Programação de Aplicativos
BLE	*Bluetooth Low Energy* / *Bluetooth* de Baixa Energia
JSR 259	*Java Specification Request 259* / Requisição de especificação Java 259
MANET	*Mobile Ad-hoc Network* / Rede Ad-hoc Móvel
P2P	*Peer to Peer* / Pessoa para Pessoa
PSN	*Pocket Switched Network* / Rede Comutada de Bolso
TCP	*Transmission Control Protocol* / Protocolo de Controle de Transmissão
UDP	*User Datagram Protocol* / Protocolo de Datagrama do Usuário
VANET	*Vehicular Ad-hoc Network* / Rede Ad-hoc Veicular

SUMÁRIO

1 INTRODUÇÃO 11
2 PROCEDIMENTOS METODOLÓGICOS 14
2.1 Estudo da área 14
2.2 Busca por tecnologias semelhantes 14
2.3 Desenvolvimento da proposta 14
2.4 Teste das melhorias 14
2.5 Cronograma 15
3 FUNDAMENTAÇÃO TEÓRICA 16
3.1 Rede Ad-Hoc 16
3.1.1 MANET 17
3.1.2 Pocket Switched Networks 18
3.2 Padrões de comunicação 18
3.2.1 Request-reply 19
3.2.2 Publish-subscribe 19
3.2.3 Pipeline 20
3.3 Métrica de agilidade de desenvolvimento 20
4 TRABALHOS RELACIONADOS 22
4.1 JSR 259 22
4.2 ZMQ 23
4.3 Nearby Connections 24
5 PROPOSTA 26
5.1 Representação da solução 27
5.1.1 Diagrama de classes completo 28
5.1.2 Diagrama de classes - Primeira parte 29
5.1.3 Diagrama de classes - Segunda parte 29
5.2 Como utilizar a melhoria 30
5.2.1 Dependências e permissões necessárias para o correto funcionamento da API 30
5.2.2 Manuseando os objetos da melhoria 32
5.3 Código fonte 32

6 TESTES E RESULTADOS 33
6.1 Aplicação de testes 33
6.2 Teste das regras de cada padrão 35
6.2.1 Teste 1 - Conexão e desconexão com comportamentos iguais 35
6.2.2 Teste 2 - Conexão e desconexão com cruzamento de comportamentos entre padrões 36
6.2.3 Teste 3 - Conexão e desconexão de comportamentos incompatíveis no mesmo padrão 37
6.2.4 Teste 4 - Conexão e desconexão de comportamentos compatíveis 38
6.2.5 Teste 5 - Envio e recebimento de dados 38
6.2.6 Teste 6 - Subscrição de serviço 39
6.2.7 Teste 7 - Confirmação de conclusão de processamento 39
6.3 Testes de processamento e armazenamento 39
6.3.1 Estado do dispositivo antes de iniciar as aplicações de teste 40
6.3.2 Estado do dispositivo durante o uso das aplicações de teste 42
6.4 Comparação de agilidade de desenvolvimento entre as aplicações A e B . 44
6.4.1 Agilidade de desenvolvimento da aplicação A 44
6.4.2 Agilidade de desenvolvimento da aplicação B 44
7 CONCLUSÕES E TRABALHOS FUTUROS 45
REFERÊNCIAS 46

1 INTRODUÇÃO

A internet nem sempre é o meio mais adequado de comunicação para todos os tipos de aplicações(HUI *et al.*, 2005a). Um bom exemplo disso são as aplicações para dispositivos móveis que usam o contexto geográfico de proximidade. Estas aplicações normalmente fazem uso unicamente da internet como meio para troca de dados. Esta dependência pode ser um problema em ambientes onde não há pontos de acesso. Outro ponto a se observar nestas aplicações, é que em determinadas situações o destino pode estar demasiadamente próximo, a ponto de se conseguir trocar dados localmente, podendo evitar o uso da infraestrutura da internet.

Para diminuir esta dependência com a internet e poder trocar dados localmente, diversos problemas desafiadores precisam ser resolvidos por estas aplicações. Esses problemas, por exemplo, nomeação, roteamento de dados e segurança já foram resolvidos pela internet(HUI *et al.*, 2005a). Porém, as aplicações citadas anteriormente podem enfrentar outro tipo de problema inerente ao contexto móvel, como: a mudança constante de topologia da rede, além de outros(KUROSE; ROSS, 2013).

As redes Ad-Hoc são ideais para o contexto anteriormente citado, mais precisamente as *Mobile Ad-hoc Network* / Rede Ad-hoc Móvel (*MANET*) por se tratar de dispositivos móveis. Porém existe um tipo de *MANET* mais específico, como é o caso da *Pocket Switched Network* / Rede Comutada de Bolso (*PSN*), sendo ainda mais aderente ao contexto citado anteriormente, onde origem e destino estão próximos. Esta maior aderência existe, também, devido ao uso de conexões locais e globais como meios de transmissão de dados, além do uso da mobilidade humana. Este tipo de rede pode ser aplicada em alguns cenários, como: prevenção de desastres naturais, trocas de mensagem, etc (HUI *et al.*, 2005a).

Um dos pontos centrais para o correto funcionamento de uma *PSN* é a comunicação via interface sem fio entre dois dispositivos móveis. Porém, fazer uso direto destas interfaces pode ser um problema a mais a ser gerenciado pela aplicação, visto a complexidade adicionada ao usar diretamente o *Bluetooth* ou o *Wi-Fi*. O uso direto da interface de *Bluetooth*, por exemplo, implica em fazer pareamento manual entre dispositivos, diminuindo a usabilidade das aplicações resultantes. Estes problemas podem ser abstraídos ao utilizar uma *Application Programming Interface* / Interface de Programação de Aplicativos (*API*), como a *Nearby Connections* (GOOGLE, 2017a), para gerenciar esta parte do trabalho. A *API* em questão, gerencia as interfaces sem fio do dispositivo para prover serviços de comunicação em um nível mais alto de abstração para as aplicações.

Apesar de a *Nearby Connections* gerenciar toda a complexidade de conexão e de troca de mensagens entre os dispositivos móveis, seu serviço é provido baseando-se em uma arquitetura *Peer to Peer* / Pessoa para Pessoa (*P2P*) muito abstrata e pouco semântica, disponibilizando somente estratégias de conexão M-para-N, 1-para-1, 1-para-N (GOOGLE, 2017c). Deste modo, nesse ponto específico, a *API* peca em fornecer meios mais semânticos de conexão limitando a qualidade do serviço disponibilizado ao programador, já que repassa para ele a responsabilidade de pensar a topologia em seus mínimos detalhes. Isso afeta diretamente a complexidade da construção da aplicação e sua qualidade.

Observando o uso de sistemas de mensagens como o ZMQ (HINTJENS, 2010), percebe-se a facilidade inserida no desenvolvimento, quando é utilizada uma *API* que permite definir qual será o papel de cada aplicação ou dispositivo. Com o uso de padrões, diversos aspectos podem ser abstraídos pelo desenvolvedor, resultando em aplicações mais simples de se desenvolver e manter, além de serem mais confiáveis por fazerem uso de uma solução amplamente utilizada e testada. Aplicar alguns dos padrões de comunicação usados no ZMQ, sobre a *API Nearby Connections* pode ser uma forma de padronizar os serviços dessa *API* e facilitar sua aplicabilidade em contextos como os abrangidos pelas *PSN*.

Diante disso, o objetivo central deste trabalho é aprimorar a *Nearby Connections* com padrões de comunicação, adaptando-a para que seja capaz de atribuir papéis para cada dispositivo, tornando-a mais semântica. Em consequência disso, facilitar sua empregabilidade em aplicações mobile contidas no contexto das *PSN*. Estes objetivos podem ser divididos em dois pontos específicos:

- Implementar uma fachada que irá prover ou restringir serviços da *Nearby Connections* dependendo do papel escolhido pela aplicação;
- Implementar os padrões de comunicação request-reply, publish-subscribe e Pipeline na *API*.

A adaptação proposta neste trabalho contribuirá com a comunidade, visto que a *API* é um projeto público. Esta adaptação pode diminuir os custos de produção de aplicações além de diminuir a curva de aprendizado. Vendo do ponto de vista da inovação, em caso de sucesso, este trabalho pode facilitar a criação de novas aplicações móveis, face a semântica introduzida.

Existem diversas tecnologias capazes de prover comunicação por interface sem fio, uma delas surgiu em resposta ao *Java Specification Request 259* / Requisição de especificação Java 259 (*JSR 259*). A *API* foi proposta pelo *Ad Hoc Networking API Expert Group* / Grupo

de Especialistas em API para Redes Ad-hoc (*AHNA-EG*) utilizando a linguagem Java(BENQ MOBILE GMBH CO OHG, 2006). A *API* proposta permite que os desenvolvedores da *API* e desenvolvedores de aplicações possam implementar um ou mais módulos protocolos para serem usados na *API* de forma mais simples devido ao emprego de interfaces bem definidas, porém se trata de uma tecnologia antiga, tendo sido lançada em 2006(BENQ MOBILE GMBH CO OHG, 2006). Diferindo disso, o ZMQ gerencia a complexidade de roteamento de mensagens, porém não provê serviços de conexão local ou ad-hoc, semelhante aos providos pela Nearby Connections e pela *JSR 259*.

Para a fundamentação conceitual deste trabalho, foi feito um levantamento bibliográfico em sistemas de busca da comunidade acadêmica, como o IEEE Explorer e ACM Digital Library. Além disso, foi realizado uma pesquisa documental sobre o ZMQ e a *Nearby Connections* para fundamentação e embasamento nas tecnologias abordadas neste trabalho. As fontes de informação utilizadas nessa pesquisa documental foram os documentos disponíveis online nas páginas oficiais de cada ferramenta e livros relacionados. Os principais autores estudados foram Pan Hui, Faruk Akgul e Augustin Chaintreau.

Este trabalho está dividido em introdução, procedimentos metodológicos, fundamentação teórica, trabalhos relacionados, proposta e testes e resultados. No Capítulo 2 será detalhado os procedimentos metodológicos realizados nesse trabalho. Durante o Capítulo 3, serão abordados os conceitos de redes onde a Nearby Connections pode ser aplicada, como: *Ad hoc*, *MANET*'s e *PSN*. No capitulo 4, será abordado as *API*'s disponíveis para criação de redes Ad-hoc e gerenciamento de mensagens. No capítulo 5, será apresentado a melhoria proposta de modo detalhado e por fim, no capítulo 6 será apresentado os testes realizados na melhoria proposta e os resultados obtidos a partir desses testes.

2 PROCEDIMENTOS METODOLÓGICOS

Cada seção desse capítulo representa uma etapa realizada no trabalho proposto, descrevendo desde pesquisas iniciais até os procedimentos efetuados para obter os resultados finais.

2.1 Estudo da área

Para esse trabalho, foi adotado a metodologia de pesquisa exploratória para levantamento dos conceitos de redes ao qual a proposta desse trabalho está inserida, como: redes Ad-Hoc, *MANET* e *PSN*. A pesquisa por esses termos foi realizada em livros e em acervos virtuais da comunidade acadêmica, como: IEEE Explorer, Google Acadêmico, ACM Digital Library, etc.

2.2 Busca por tecnologias semelhantes

Foram realizadas buscas por tecnologias semelhantes a Nearby Connections com a finalidade de averiguar pontos positivos que possam melhorar a Nearby Connections. Nessas buscas foram utilizados palavras chave relacionadas ao contexto desse trabalho, como: *API*, *Library*, Ad-Hoc, *Pocket Switched Network*, *Service*, *Connection*, *Transmission*, *off-line* etc. A partir dessas pesquisas, foram encontrados o ZMQ com seus padrões de comunicação e a *JSR 259*, que provê serviços Ad-Hoc em linguagem Java. Para maior entendimento das tecnologias encontradas, incluindo a Nearby Connections, foram realizadas pesquisas documentais nos seus respectivos sites de documentação oficial.

2.3 Desenvolvimento da proposta

A solução proposta neste trabalho será construída em linguagem Java, até então, linguagem nativa para desenvolvimento de aplicativos para plataforma Android.

2.4 Teste das melhorias

Após a realização das melhorias, será desenvolvida uma aplicação com interface gráfica que permita o teste das funcionalidades inseridas na melhoria, permitindo a escolha do papel que o dispositivo irá possuir e executar todas as ações relacionadas a este papel. A partir

disso, será averiguado o correto funcionamento das melhorias feitas, ou seja, averiguado se a aplicação está funcionando como espera-se dado o papel/comportamento escolhido por esta. Também serão implementadas outras duas aplicações com o propósito de comparar o uso de processamento e memória entre uma aplicação que utilize diretamente a Nearby Connections e outra aplicação que utilize a melhoria proposta. Por fim, será aplicada, nessas duas aplicações, a métrica citada por Mikio Ikoma *et al.* (2009) em seu trabalho, com a finalidade de verificar se é mais simples e ágil utilizar a melhoria proposta.

2.5 Cronograma

Os prazos para elaboração e realização das atividades passadas e futuras desse trabalho estão descritos na Figura 1.

Figura 1 – Cronograma do trabalho.

	2019					2020				
ATIVIDADES	**8**	**9**	**10**	**11**	**12**	**1**	**2**	**3**	**4**	**5**
Definição da proposta	X									
Revisão bibliográfica	X	X								
Pesquisa por tecnologias semelhantes		X	X							
Estudo de aplicação da proposta		X	X	X						
Escrita do TCC 1	X	X	X	X						
Defesa do TCC 1				X						
Implementação das melhorias					X	X	X	X		
Implementação da aplicação de teste							X	X	X	
Teste das funcionalidades implementadas								X	X	
Escrita do TCC 2						X	X	X	X	
Defesa do trabalho final										X

Fonte: Autor.

3 FUNDAMENTAÇÃO TEÓRICA

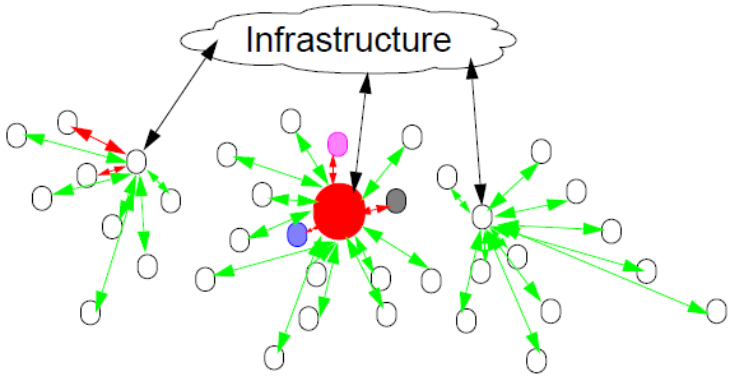
Diante do contexto que este trabalho está situado, é necessária a compreensão de conceitos de redes de computadores presentes na literatura que circundam e interferem no modo que as tecnologias abordadas neste trabalho funcionam. De modo específico, os tipos de redes que utilizam troca de dados localmente entre dispositivos móveis. Dessa forma, faz-se necessário um nível de entendimento de determinados tipos de redes, como: as redes Ad-Hoc, as *MANET* e as *PSN*. Também são objetos de estudo e análise nesse capítulo os padrões de comunicação request-reply, publish-subscribe, pipeline e a métrica utilizada para medir a agilidade de desenvolvimento de projetos.

3.1 Rede Ad-Hoc

O conceito de rede Ad-Hoc surgiu por volta da década de setenta com o intuito de assegurar a troca de dados entre computadores por meio da comutação de pacotes em ambientes onde, normalmente, não existe infraestrutura fixa e previsível(MURTHY; MANOJ, 2004). Sendo classificado de acordo com o ambiente ao qual está aplicado, podendo ser: *Vehicular Ad-hoc Network* / Rede Ad-hoc Veicular (*VANET*), *MANET*'s, entre outras(KUROSE; ROSS, 2013).

As redes convencionais que utilizamos, funcionam sobre uma topologia bem definida com dispositivos de acesso centralizadores como: roteadores e *switchs*. Podemos observar este funcionamento na Figura 2.

Figura 2 – Diagrama de uma rede convencional.

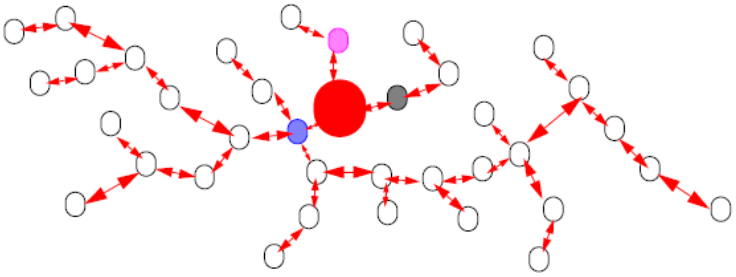


Fonte: (LIN *et al.*, 2001).

Já uma rede Ad-Hoc é projetada para permitir comunicação de forma descentralizada, onde os nós que a compõem estão conectados entre si e são responsáveis por rotear os dados que trafegam por ela (KUROSE; ROSS, 2013). Deste modo, a falha de um ou mais nós não inviabiliza

completamente a rede e o tráfego de dados(KUROSE; ROSS, 2013). Este funcionamento pode ser observado na Figura 3.

Figura 3 – Diagrama de uma rede Ad-Hoc.

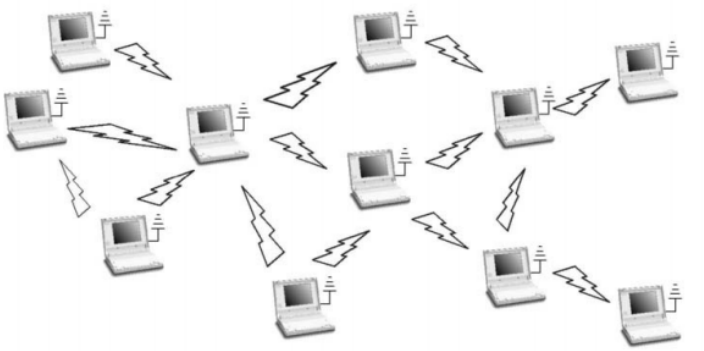


Fonte: (LIN *et al.*, 2001).

3.1.1 MANET

Nesse tipo de rede Ad-Hoc os dispositivos que a compõem podem se locomover, podendo perder conexão com determinados dispositivos e criar conexões com outros, gerando complexidade no momento do roteamento de dados(KURKOWSKI *et al.*, 2005). Para lidar com esse problema, uma *MANET* utiliza a abordagem multi-hop para transferir dados de uma origem para um destino, repassa os dados entre dispositivos próximos e assim por diante, até que cheguem ao seu destino(KUROSE; ROSS, 2013). Para realizar esse repasse, são utilizadas interfaces sem fio como meio de troca de dados. O conceito de *MANET* pode ser observado na Figura 4.

Figura 4 – Exemplo de *MANET*.



Fonte: (TSENG *et al.*, 2003).

3.1.2 Pocket Switched Networks

Uma *PSN* se assemelha bastante a uma rede ad-hoc no quesito de descentralização da rede e de sua imprevisibilidade topológica. Porém, difere no fato de que em uma *PSN*, conexões a redes globais, como a internet, e conexões locais devem ser utilizadas para transferência dados até seu destino(HUI *et al.*, 2005a).

A *PSN* também utiliza a abordagem multi-hop, mencionada anteriormente, diferenciando-se das abordagens normalmente utilizadas nas redes Ad-Hoc, onde um caminho fim-a-fim entre o dispositivo origem e o dispositivo destino é construído para que haja a transmissão de um dado(HUI *et al.*, 2005b). Além disso, uma *PSN* se diferencia em suas premissas, que definem o comportamento deste tipo de rede. Segue as premissas, propostas por Hui *et al.* (2005a), que diferenciam uma *Pocket Switched Network*:

- Usuários de dispositivos móveis podem possuir um ou mais dispositivos, contribuindo com suas capacidades de armazenamento;
- Levar dados de um lugar à outro usufruindo da mobilidade destes dispositivos;
- Os dispositivos móveis possuem interfaces sem fio, podendo fazer uso delas para trocar dados com dispositivos próximos;
- Dispositivos móveis podem possuir conexão com mais de um tipo de rede global, como por exemplo, a internet;
- Para que haja mais ocasiões de transferência de dados, deve-se utilizar tanto conexões locais, quanto conexões com redes globais.

A mobilidade dos dispositivos móveis é um ponto chave neste tipo de rede podendo aumentar as oportunidades de transferência de dados e por consequência a capacidade de transferência da rede como um todo. Porém, pode aumentar a dificuldade de comunicação entre dispositivos, visto a alta variabilidade topológica dos dispositivos na rede, tornando a acessibilidade dos dispositivos bastante variável (HUI *et al.*, 2005b).

3.2 Padrões de comunicação

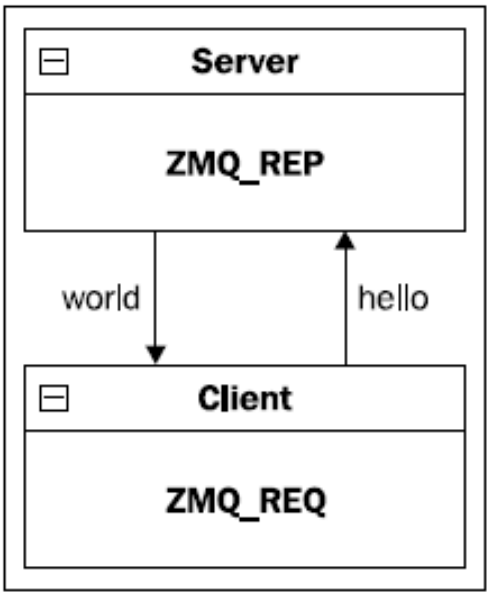
Os padrões de comunicação apresentado nas subseções a seguir são padrões já testados e utilizados em ferramentas voltadas para comunicação via internet, como por exemplo o ZMQ. Devido a isso e a sua simplicidade, serão utilizados mais adiante como parte da melhoria proposta. A escolha dos três padrões abaixo se dão devido a dois fatores principais: tempo

disponível para realização do trabalho e a capacidade de aplicá-los em diversos cenários por conta do grande uso de soluções desse tipo no ambiente computacional, como por exemplo a estrutura de cliente-servidor.

3.2.1 Request-reply

O padrão request-reply pode ser utilizado quando há a necessidade de se ter uma relação semelhante a uma arquitetura cliente-servidor, onde um sistema cliente requisita dados a um sistema servidor, e então espera o recebimento do que foi requisitado(HINTJENS, 2010). Quanto ao sistema servidor, sua responsabilidade é ouvir e responder estas requisições(HINTJENS, 2010). Esta relação é melhor vista na Figura 5.

Figura 5 – Diagrama do padrão request-reply.



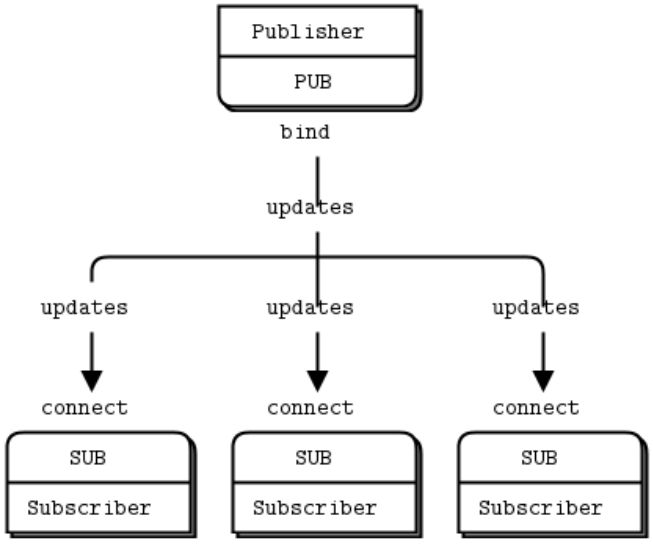
Fonte: (AKGUL, 2013).

3.2.2 Publish-subscribe

O padrão publish-subscribe também será abordado neste trabalho. Este padrão é específico para situações de relação 1-N entre as aplicações, com somente um sentido de comunicação(HINTJENS, 2010). Utilizando a analogia feita por Akgul (2013) em seu livro, o comportamento deste padrão se assemelha aos canais de TV e estações de rádio, onde uma aplicação é responsável por produzir determinado tipo de conteúdo e enviar para as aplicações

previamente registradas para receber esta produção. Vale ressaltar que a produção de conteúdo independe da existência de aplicações registradas para receber este conteúdo. Pode-se observar o funcionamento deste padrão na Figura 6.

Figura 6 – Diagrama do padrão publish-subscribe.



Fonte: (HINTJENS, 2010).

3.2.3 Pipeline

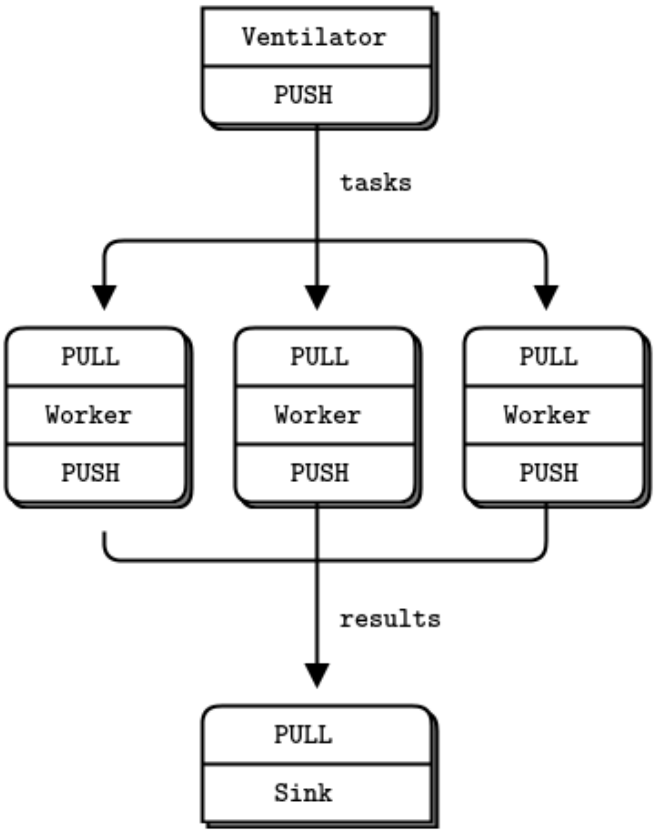
O padrão pipeline será o terceiro padrão de comunicação abordado neste trabalho. Este padrão é bastante útil em situações de processamento paralelo entre diversas máquinas, possuindo três papéis, o papel do Ventilator que é responsável por distribuir dados para os Workers, que são responsáveis por realizar algum trabalho computacional sobre o dado recebido do Ventilator e então repassar o resultado desse trabalho para o Sync, que então irá juntar esses resultados (HINTJENS, 2010).Pode-se observar o funcionamento deste padrão na Figura 7.

3.3 Métrica de agilidade de desenvolvimento

De acordo com Mikio Ikoma *et al.* (2009), a agilidade de desenvolvimento de um software pode ser medida pela fórmula **E = V / U**, onde:

- E - É a eficiência de desenvolvimento de um software;
- V - É o total de entregas para um determinado período;
- U - É a quantidade média de entregas intermediárias.

Figura 7 – Diagrama do padrão pipeline.



Fonte: (HINTJENS, 2010).

4 TRABALHOS RELACIONADOS

Vista a particularidade da proposta, tecnologias e ferramentas de propósito semelhante foram os objetos da seleção dentro dos resultados retornados na busca por trabalhos relacionados. Como resultado dessa seleção, os trabalhos semelhantes são *JSR 259* e o ZMQ. Além disso, será abordado nesse capítulo a Nearby Connections, *API* que será o objeto nesse trabalho.

4.1 JSR 259

Esta *JSR* tem como objetivo prover meios para que desenvolvedores possam criar redes Ad-hoc, de modo mais simples e com menos complexidade, utilizando a linguagem de programação Java(BENQ MOBILE GMBH CO OHG, 2006). Assemelha-se a proposta desse trabalho pois introduz e permite a introdução de novos módulos de protocolos a serem utilizados pelas aplicações como pode ser visto no escopo de BenQ Mobile GmbH Co OHG (2006). Nesse caso, na melhoria proposta nesse trabalho, são introduzidos padrões de comunicação ao invés de protocolos.

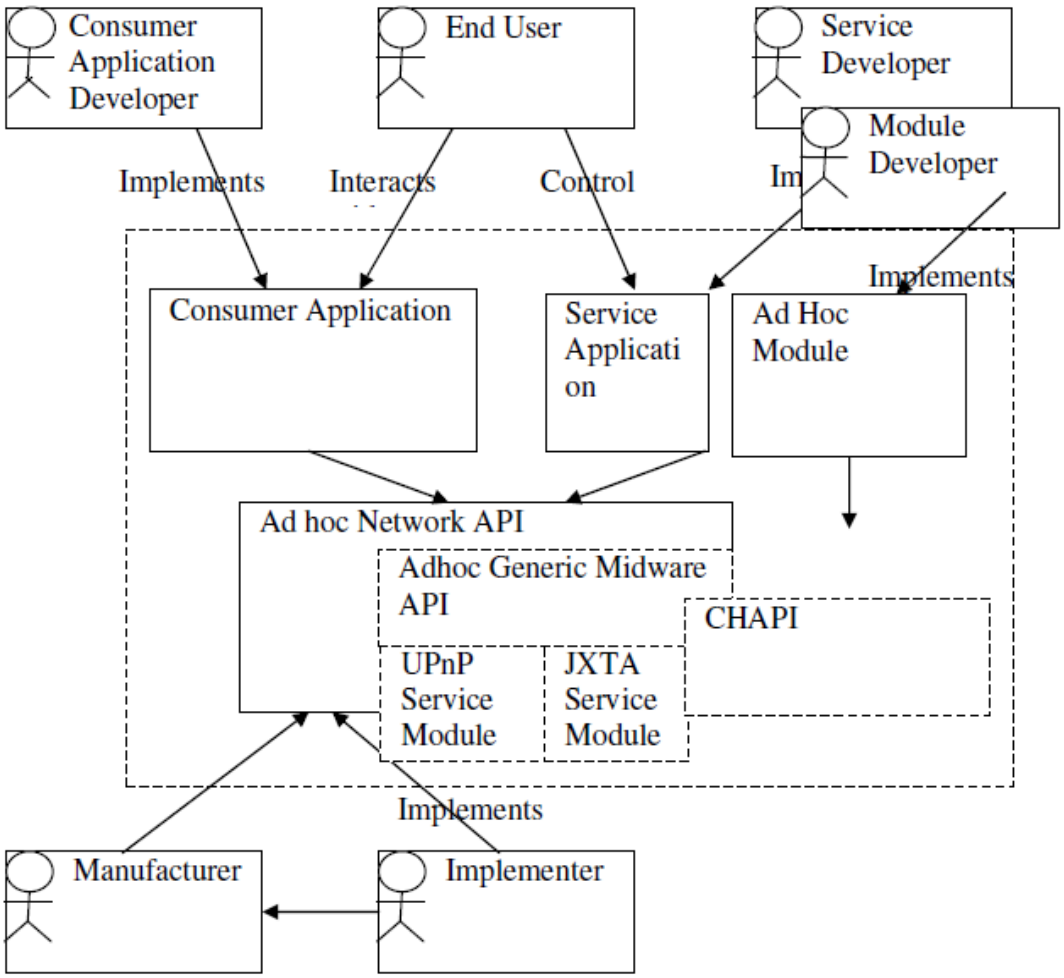
Sua arquitetura está principalmente dividida nos seguintes módulos: *Consumer Application*, *Service Application*, *Ad Hoc Module*, *Ad Hoc Network API*(BENQ MOBILE GMBH CO OHG, 2006).

O modo ao qual esses módulos estão relacionados e quais atores interagem com cada um deles pode ser melhor observado na Figura 8.

Como descrito em BenQ Mobile GmbH Co OHG (2006), os principais módulos e atores quem compõem a *JSR 259* apresentam as seguintes características:

1. ***Consumer application*** – Contém as regras de negócio e é responsável por utilizar a *API* para acessar as redes Ad-Hoc. Também é responsável por buscar e consumir os serviços prestados pela *API*.
2. ***Service Application*** – Contém o serviço em si, sendo responsável por utilizar a *API* para tornar seu serviço público e prove-lo quando for requisitado.
3. ***Consumer Application Developer*** – Este ator é responsável por desenvolver a aplicação que será utilizada pelo usuário final.
4. ***Service Developer*** – Este ator é responsável por desenvolver o serviço e torná-lo acessível por meio da *API JSR*.

Figura 8 – Diagrama arquitetural da *API* proposta na *JSR 259*.



Fonte: (BENQ MOBILE GMBH CO OHG, 2006).

4.2 ZMQ

ZMQ é um *framework* para ambientes de processamento concorrente, sendo atualmente utilizado pela Microsoft, Samsumg e pelo Spotify. Este *framework* permite a troca de mensagens entre sistemas distribuídos e aplicações simultâneas de diversas formas, como: *Transmission Control Protocol* / Protocolo de Controle de Transmissão (*TCP*), entre processos, *multicast*, etc(HINTJENS, 2010). Além disso, permite o processamento assíncrono de mensagens. Seus serviços são providos por meio de padrões de comunicação, o que o torna mais semântico comparando-se ao uso direto de *sockets TCP* e *User Datagram Protocol* / Protocolo de Datagrama do Usuário (*UDP*)(HINTJENS, 2010). Assemelha-se a este trabalho no quesito de envio de mensagens e pelo uso dos padrões de projeto request-reply e publish-subscribe.

4.3 Nearby Connections

A Nearby Connections é uma *API* projetada pelo Google para facilitar a conexão e a troca de dados entre dispositivos móveis por meio de conexões encriptadas e com baixa latência, estando localizada no pacote com.google.android.gms.nearby.connection (GOOGLE, 2017a). Pode ser aplicada para diversas finalidades, como: transferência *off-line* de arquivos, jogos multi-telas etc (GOOGLE, 2017a). A *API* funciona sobre o *Bluetooth*, *Bluetooth Low Energy* / *Bluetooth* de Baixa Energia (*BLE*) e o Wi-Fi, utilizando-se dos pontos fortes de cada uma destas tecnologias para prover um serviço mais simples e rápido de utilizar(ANTONIOLI *et al.*, 2019).

Para que uma conexão seja estabelecida, é necessário que haja um dispositivo se anunciando e um outro dispositivo descobrindo(ANTONIOLI *et al.*, 2019). Para realizar estas duas ações, as aplicações devem escolher um dos modos de conexão disponíveis(GOOGLE, 2017c). Podendo ser:

- P2P CLUSTER - Abordagem de conexão M para N dispositivos;
- P2P STAR - Abordagem de conexão 1 para N dispositivos;
- P2P POINT TO POINT - Abordagem de conexão 1 para 1 dispositivos.

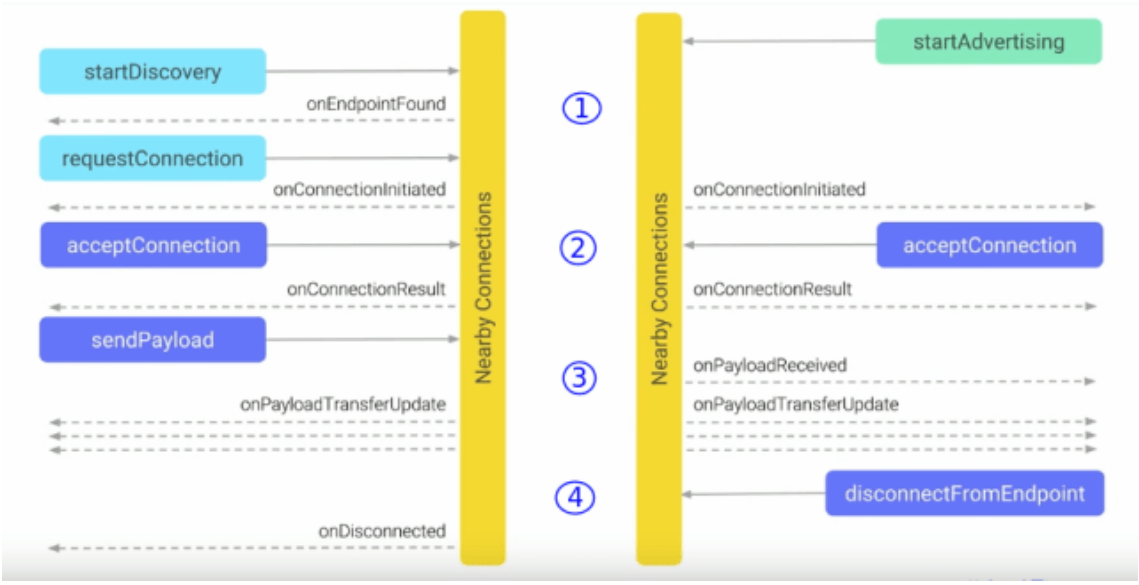
Para que de fato a conexão seja estabelecida, diversos métodos são acionados em sequência para que ambos os dispositivos estejam em acordo com o estado da conexão (GOOGLE, 2017a). Esta sequência acontece até o passo em que ambos os dispositivos recebem o resultado final da tentativa de conexão pelo método 'OnConnectionResult', onde, a partir de então, a conexão pode estar estabelecida.

A sequência mencionada anteriormente pode ser melhor observada na Figura 9.

A *API* realiza a troca de dados por meio de objetos 'Payload', que podem levar três tipos de dados diferentes(GOOGLE, 2017a). Sendo eles:

- Bytes - Vetores de Bytes de tamanho 32k;
- Arquivo - Arquivos de qualquer tamanho;
- Stream - Arquivos que não possuem um tamanho final definido.

Figura 9 – Fluxo da Nearby Connections.

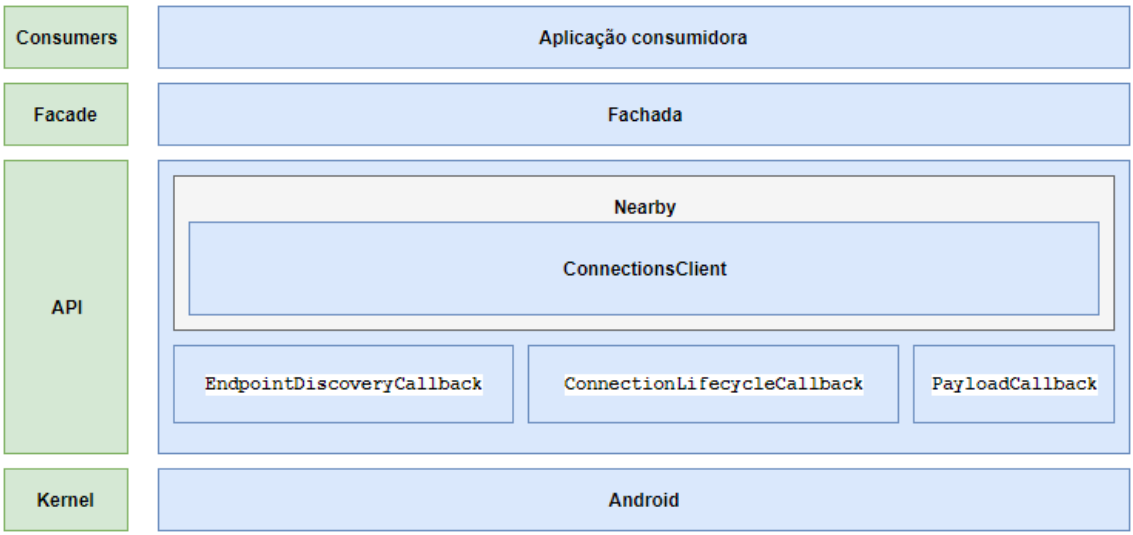


Fonte: (BRIJESH, 2018).

5 PROPOSTA

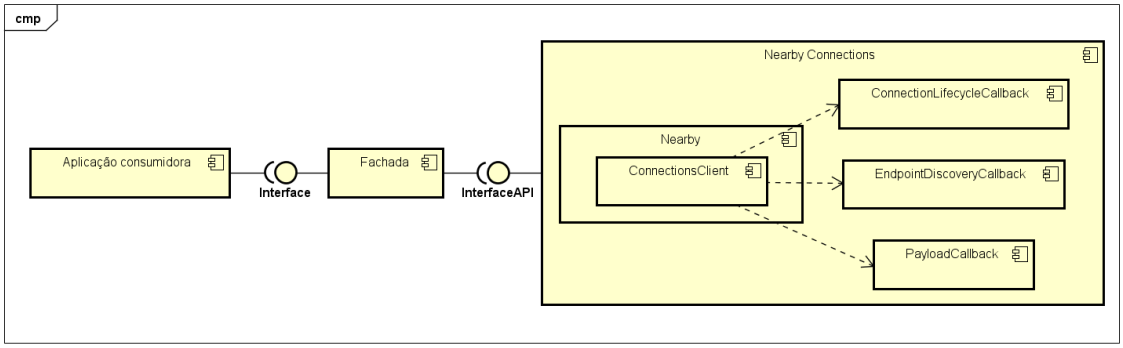
A melhoria proposta consiste em introduzir um objeto fachada sobre a *Nearby Connections* de modo a introduzir na *API*, os papéis *Requester*, *Replyer*, *Publisher*, *Subscriber*, *Ventilator*, *Worker* e *Sync*. Desse modo, as aplicações que fizerem uso dessa fachada poderão utilizar estes papéis, o que indiretamente implica no uso dos padrões de comunicação citados anteriormente. Dessa forma, essas aplicações farão uso de funcionalidades com maior nível de abstração, facilitando o desenvolvimento de *software*. Essa melhoria pode ser melhor observada nas Figuras 10 e 11.

Figura 10 – Arquitetura proposta.



Fonte: Autor.

Figura 11 – Diagrama de componentes da proposta.



Fonte: Autor.

Cada componente na melhoria proposta exercerá as seguintes funções:

- **Fachada** - Fará a comunicação entre as classes da aplicação consumidora e a *Nearby Connections*, restringindo ou liberando serviços específicos de modo a obedecer o pa-

pel(*Requester*, *Replyer*, *Publisher*, *Subscriber*, *Ventilator*, *Worker* ou *Sync*) escolhido pela aplicação. Dessa forma, torna-se mais simples restringir ou liberar serviços, sem haver modificação no código fonte da *API*. Os demais componentes pertencem a *API* e já estão implementados.

- **Nearby** - Objeto de fachada da *API*.
- **ConnectionsClient** - Responsável por manter as conexões, realizar o envio de objetos de dados, repassar requisições de conexão do EndpointDiscoveryCallBack para o ConnectionLifecycleCallback, iniciar procedimento de anúncio, iniciar procedimento de descoberta, entre outras.
- **EndpointDiscoveryCallBack** - Gerencia o processo de descoberta de dispositivos, tratando possíveis exceções e requisitando ao ConnectionsClient repasse a requisição de conexão para o ConnectionLifecycleCallback, caso a descoberta tenha sido bem-sucedida. É utilizado pelo componente ConnectionsClient ao iniciar uma descoberta.
- **ConnectionLifecycleCallback** - Gerencia os estágios das conexões entre os dispositivos após receber uma requisição indireta do EndpointDiscoveryCallBack. Sua principal responsabilidade é sincronizar o processo de conexão entre ambos os dispositivos de modo que estabeleçam uma conexão ou que ambos cheguem a mesma conclusão sobre a tentativa de conexão. É utilizado pelo componente ConnectionsClient ao iniciar um anúncio e também quando um dispositivo é encontrado no processo de descoberta.
- **PayloadCallBack** - Gerencia o recebimento de objetos de dados do tipo Payload após uma conexão ser estabelecida. É utilizado como insumo pelo componente ConnectionsClient quando uma conexão está sendo aceita, dessa forma o PayloadCallBack passa a gerenciar o recebimento de dados dessa conexão recém estabelecida.

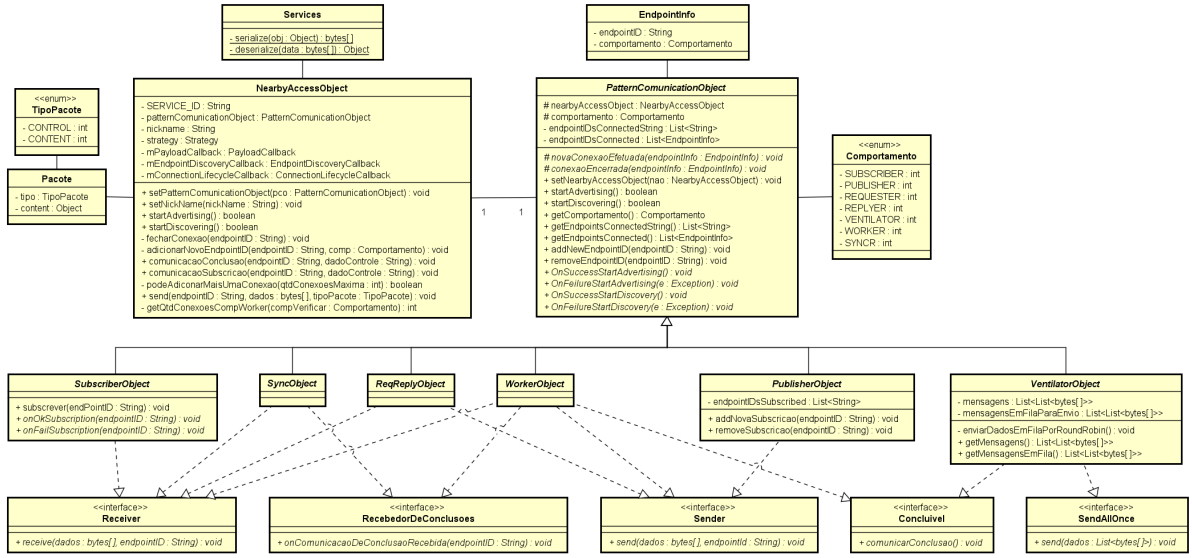
5.1 Representação da solução

Nessa seção será apresentada uma visão computacional de como a solução está implementada, para essa finalidade a melhoria será apresentada em forma de diagrama de classes, descrevendo os objetos que serão utilizados na melhoria assim como seus atributos e seus métodos.

5.1.1 Diagrama de classes completo

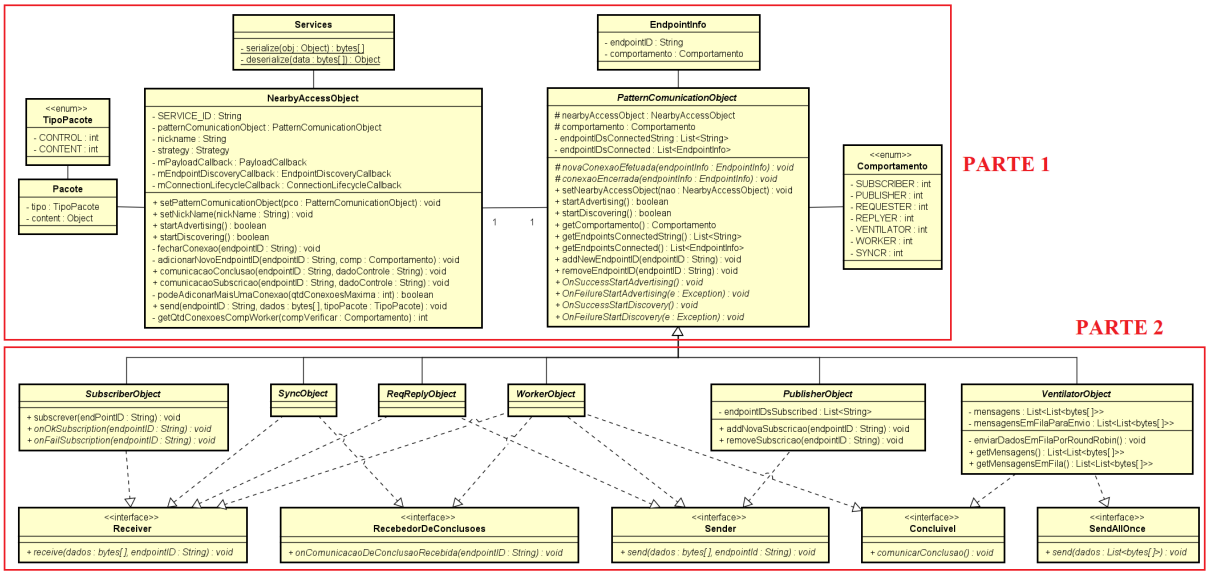
A Figura 12 apresenta o diagrama em uma visão completa, porém para melhorar a visualização do mesmo, este diagrama será dividido em duas partes nas subseções seguintes como demonstrado na Figura 13. Os diagramas mostrados abaixo podem ser consultados diretamente pelo Link do repositório.

Figura 12 – Diagrama de classes da proposta.



Fonte: Autor.

Figura 13 – Divisões no diagrama de classes da proposta.



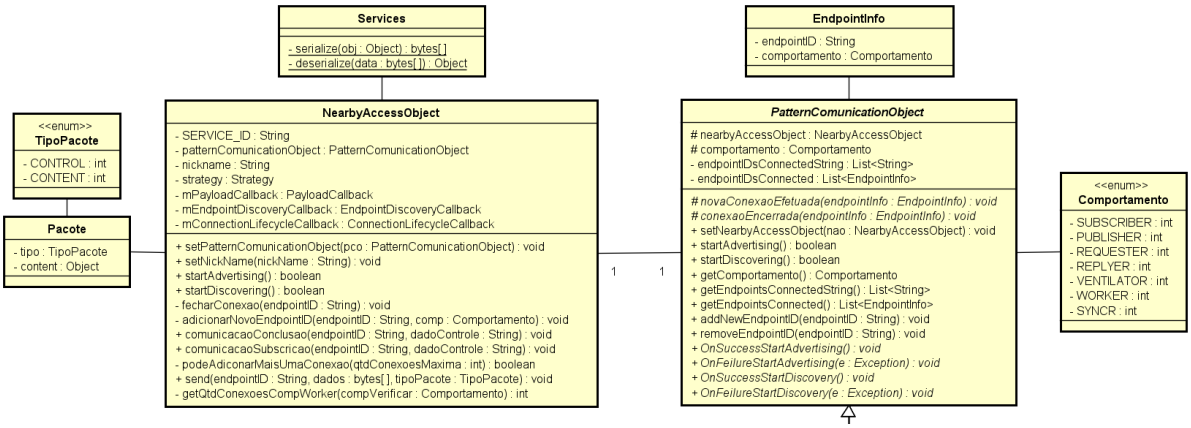
Fonte: Autor.

5.1.2 Diagrama de classes - Primeira parte

A primeira parte, demonstrada pela Figura 14, consiste-se basicamente das duas principais classes da melhoria proposta(*NearbyAccesObject* e *PatternComunnicationObject*) e algumas classes e enums auxiliares.

A classe *NearbyAccesObjet* é responsável por intermediar e gerenciar as configurações de acesso e comunicação da *Nearby Connections* com a classe *PatternComunnicationObject*. Já a classe *PatternComunnicationObject* é responsável por agrupar atributos e métodos em comum a todos os objetos do padrão de comunicação que serão especificados na próxima seção, além de ser a responsável por requisitar os serviços através da classe *NearbyAccesObjet*. As classes Pacote, *Services* e *EndpointInfo* são utilizadas para prestar serviços básicos ou agrupar e estruturar dados.

Figura 14 – Primeira parte da divisão no diagrama de classes da proposta.



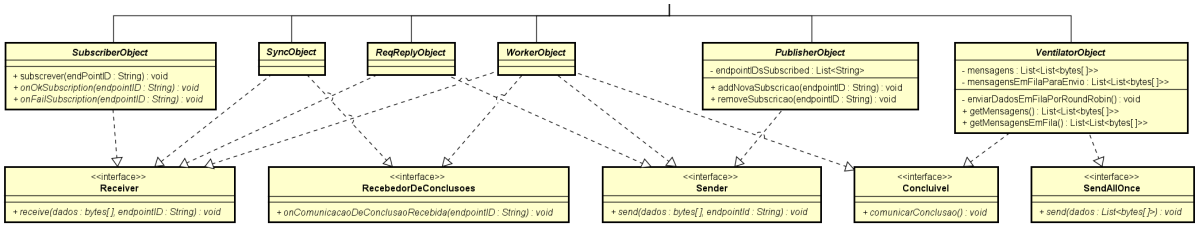
Fonte: Autor.

5.1.3 Diagrama de classes - Segunda parte

A Segunda parte, demonstrada pela Figura 15, consiste-se nas diversas classes abstratas que representam, cada uma, um papel. Além disso, também pode-se visualizar três interfaces responsáveis por definir quais objetos podem receber ou enviar dados, o que por sua vez depende do comportamento de cada papel. As classes abstratas definidas na Figura 15 são utilizadas para serem herdadas por uma classe criada pelo desenvolvedor usuário dessa melhoria, desse modo o mesmo será obrigado a definir uma programação para os métodos abstratos listados. Cada classe abstrata está implementada de modo que o desenvolvedor não necessite se atentar na comunicação com a classe *NearbyAccesObjet*, os métodos que necessitam de tal atenção já estão

previamente programados.

Figura 15 – Segunda parte da divisão no diagrama de classes da proposta.



Fonte: Autor.

5.2 Como utilizar a melhoria

Nessa seção será descrito quais procedimentos serão necessários para que a melhoria possa ser corretamente aplicada em outros projetos.

5.2.1 Dependências e permissões necessárias para o correto funcionamento da API

Para que a *API* possa funcionar corretamente é necessários que o desenvolvedor faça algumas modificações nos arquivos de Gradle do seu projeto.

A primeira dependência deve ser introduzida no gradle do projeto, como demonstrado pela **Figura 16**. Essa dependência é responsável por permitir que o projeto android consiga importar a dependência da melhoria, como veremos mais a diante. Essa dependência se torna necessária pelo fato da melhoria ter sido disponibilizada utilizando a plataforma jitpack.io.

Figura 16 – Modificação no build.gradle do projeto.

```
allprojects {
    repositories {
        google()
        jcenter()
        maven { url 'https://jitpack.io' }
    }
}
```

Fonte: Autor.

Já a segunda dependência será introduzida no gradle do referente ao app, como demonstrado na **Figura 17**. Essa dependência é responsável por importar a API Nearby Connections já com a melhoria proposta, além das permissões necessárias para que a API funcione corretamente.

Figura 17 – Modificação no build.gradle(Module: app).

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'androidx.appcompat:appcompat:1.1.0'
    implementation 'com.github.MarceloMra:NearbyFenixExample:-SNAPSHOT'
    implementation 'androidx.constraintlayout:constraintlayout:1.1.3'
    testImplementation 'junit:junit:4.12'
    androidTestImplementation 'androidx.test.ext:junit:1.1.1'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
    implementation 'androidx.recyclerview:recyclerview:1.0.0'
}
```

Fonte: Autor.

Por último, será necessário introduzir algumas linhas de código no arquivo **Manifest.xml** do projeto Android do desenvolvedor, como demonstrado na **Figura 18**. A primeira modificação será no atributo "name", que deve ser introduzido no arquivo xml sendo responsável por apontar a classe responsável por guardar, globalmente, o contexto da aplicação, para que esse contexto possa ser utilizado, tanto pelas classes do desenvolvedor, quanto pelas classes da melhoria. A segunda modificação é responsável por definir um identificador para sua aplicação, que será utilizado pela Nearby Connections.

Figura 18 – Modificações do arquivo Manifest.xml.

MainActivity.java | AndroidManifest.xml | build.gradle (:app) | build.gradle (NearbyImprovement)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.nearbyimprovement">

    <application
        android:name="com.example.nearbyfenix.improvement.GlobalApplication"
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="Nearby Improvement"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".activities.MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>

        <meta-data
            android:name="com.google.android.gms.nearby.connection.SERVICE_ID"
            android:value="@string/service_id" />
    </application>

</manifest>
```

Fonte: Autor.

5.2.2 Manuseando os objetos da melhoria

O desenvolvedor deverá criar uma classe para cada comportamento do padrão que este desejar utilizar, essas classes deverão herdar de cada respectiva classe abstrata do padrão escolhido, como por exemplo, **MyPublisherObject** como sendo a nova classe criada pelo desenvolvedor que deverá herdar da classe **PublisherObject** que já está implementada pela melhoria. Com isso o desenvolvedor poderá utilizar as funcionalidades disponíveis em **PublisherObject** e será obrigado a implementar os métodos abstratos contidos em **PublisherObject** na sua nova classe **MyPublisherObject**. Com cada classe implementada, o desenvolvedor precisará somente instanciar um objeto do tipo **NearbyAccessObject** e um outro do tipo referente a alguma classe por ele implementada, como por exemplo **MyPublisherObject**, e então adicionar a referência de um objeto no outro utilizando o método **setPatternComunicationObject()** da classe **NearbyAccessObject** e método **setNearbyAccessObject()** que estará contido em **MyPublisherObject**.

5.3 Código fonte

O condigo-fonte referente a melhoria proposta nesse trabalho se encontra disponível para consulta através da plataforma do github pelo Link do repositório ou por acesso direto a url do repositório <https://github.com/MarceloMra/NearbyFenixExample>.

6 TESTES E RESULTADOS

Nesse capítulo são apresentados os testes, seus resultados e a aplicação utilizada para realizar esses testes manuais das funcionalidades implementadas na melhoria proposta. Os testes apresentados foram realizados com dispositivos reais que possuem o sistema operacional Android. A aplicação de testes apresentada pode ser consultada pelo seguinte link do Github <https://github.com/MarceloMra/NearbyImprovement>.

6.1 Aplicação de testes

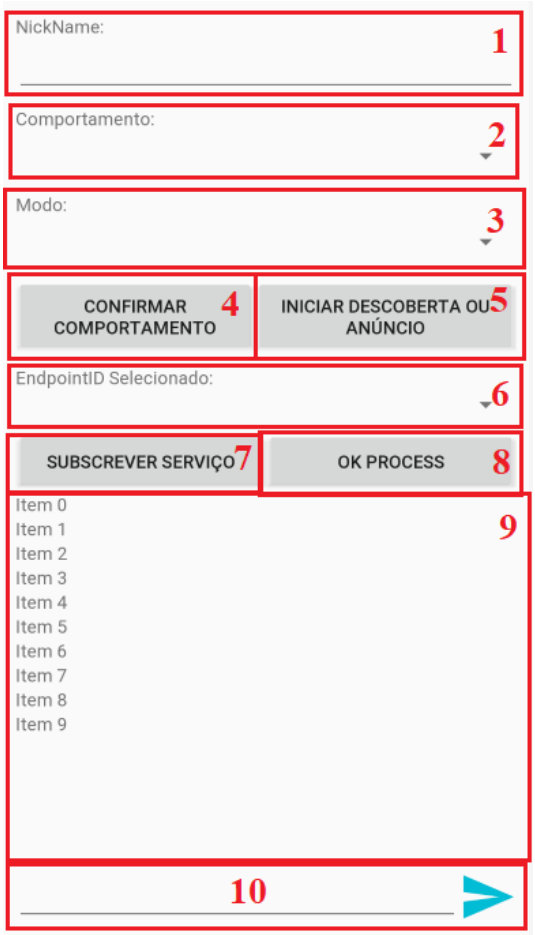
A aplicação de testes está programada para trocar dados do tipo *String* para facilitar o envio e resposta de dados durante os testes, porém a melhoria permite o envio e o recebimento de praticamente qualquer tipo de conteúdo, desde que este possa ser transformado em *bytes*, que é o tipo de dado trocado pela API. A **Figura 19** demonstra como a aplicação de teste está estruturada visualmente.

Figura 19 – Aplicação de testes.

Fonte: Autor.

Para fins descritivos, a **Figura 19** será dividida e grafada em dez componentes, como demonstrado na **Figura 20**.

Figura 20 – Componentes da aplicação de testes.



Fonte: Autor.

Cada componente gráfico grafado na **Figura 20** desempenha as seguintes funções descritas abaixo:

- **Componente 1** - Campo de texto utilizado para que o testador entre com um nickname, esse dado é requisitado e utilizado pela *API Nearby Connections*.
- **Componente 2** - Campo do tipo Spinner utilizado para que o testador escolha um dos papéis/comportamento deseja utilizar.
- **Componente 3** - Campo do tipo Spinner utilizado para que o testador escolha entre anunciar-se ou descobrir dispositivos próximos.
- **Componente 4** - Botão utilizado para confirmar o *nickname* e o papel/comportamento escolhido pelo testador.
- **Componente 5** - Botão responsável por iniciar o processo de descoberta ou anúncio.

- **Componente 6** - Campo do tipo Spinner responsável por listar os endpointIDs dos dispositivos conectados no momento e manter um desses IDs selecionado.
- **Componente 7** - Botão responsável por realizar uma requisição de subscrição de um serviço, a requisição é direcionada ao dispositivo selecionado no **Componente 6**. Esse botão somente é acessível caso o papel/comportamento escolhido no **Componente 2** seja o comportamento *Subscriber*.
- **Componente 8** - Botão responsável por enviar uma confirmação de finalização de processamento/envio de dados, a requisição é direcionada ao dispositivo selecionado no **Componente 6**. Esse botão somente é acessível caso o papel/comportamento escolhido no **Componente 2** seja o comportamento *Ventilator* ou *Worker*.
- **Componente 9** - Área de visualização de mensagens de dados enviadas e recebidas, constando a hora, o endpointID e a mensagem. Para mensagens enviadas, constará o ID do dispositivo destino, já para as recebidas constará o ID do dispositivo remetente, com exceção do papel *Publisher* que sempre envia dados para todos os dispositivos que subscreveram seu serviço.
- **Componente 10** - Área de escrita e envio de mensagens.

6.2 Teste das regras de cada padrão

Nessa sessão serão apresentados os testes realizados na melhoria. Os testes foram realizados com dispositivos reais devido a dificuldade encontrada em simular o *hardware* de Bluetooth já que o Android Emulator não possui suporte para simular esse tipo de *hardware* virtualmente (GOOGLE, 2017b). Também foi realizado uma tentativa de emulação do sistema Android utilizando o programa Virtual Box em um notebook com *hardware* de Bluetooth, porém não houve êxito devido a baixa capacidade de processamento do notebook que ocasionou diversos travamentos que impossibilitaram a continuidade da instalação do Android versão 9.0 na máquina virtual. Os testes nessa sessão serão subdivididos em subseções onde será descrito a finalidade do teste, resultado esperado e resultado obtido.

6.2.1 Teste 1 - Conexão e desconexão com comportamentos iguais

Os testes apresentados na **Figura 21** foram realizados para averiguar se os dispositivos são desconectados a partir do momento que detectam possuir o mesmo comportamento,

ou seja, comportamentos incompatíveis. Os resultados obtidos foram iguais aos resultados desejados, onde os dispositivos encerram a conexão entre si.

Figura 21 – Resultados do teste 1, comportamentos iguais.

Conexão e desconexão com comportamentos iguais			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Requester	Requester	Conexão Encerrada	Conexão Encerrada
Replyer	Replyer	Conexão Encerrada	Conexão Encerrada
Publisher	Publisher	Conexão Encerrada	Conexão Encerrada
Subscriber	Subscriber	Conexão Encerrada	Conexão Encerrada
Ventilator	Ventilator	Conexão Encerrada	Conexão Encerrada
Worker	Worker	Conexão Encerrada	Conexão Encerrada
Sync	Sync	Conexão Encerrada	Conexão Encerrada

Fonte: Autor.

6.2.2 Teste 2 - Conexão e desconexão com cruzamento de comportamentos entre padrões

Os testes apresentados na **Figura 22** e na **Figura 23** possuem a finalidade de garantir que os dispositivos com determinado comportamento só possa se conectar com outros comportamentos do mesmo padrão. Cada comportamento foi testado com todos os demais comportamentos dos outros padrões. Os resultados obtidos também foram iguais aos resultado desejados.

Figura 22 – Resultados do teste 2A, Cruzamento de comportamentos.

Conexão e desconexão - Requester com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Requester	Publisher	Conexão Encerrada	Conexão Encerrada
Requester	Subscriber	Conexão Encerrada	Conexão Encerrada
Requester	Ventilator	Conexão Encerrada	Conexão Encerrada
Requester	Worker	Conexão Encerrada	Conexão Encerrada
Requester	Sync	Conexão Encerrada	Conexão Encerrada

Conexão e desconexão - Replyer com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Replyer	Publisher	Conexão Encerrada	Conexão Encerrada
Replyer	Subscriber	Conexão Encerrada	Conexão Encerrada
Replyer	Ventilator	Conexão Encerrada	Conexão Encerrada
Replyer	Worker	Conexão Encerrada	Conexão Encerrada
Replyer	Sync	Conexão Encerrada	Conexão Encerrada

Fonte: Autor.

Figura 23 – Resultados do teste 2B, Cruzamento de comportamentos.

Conexão e desconexão - Publisher com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Publisher	Requester	Conexão Encerrada	Conexão Encerrada
Publisher	Replyer	Conexão Encerrada	Conexão Encerrada
Publisher	Ventilator	Conexão Encerrada	Conexão Encerrada
Publisher	Worker	Conexão Encerrada	Conexão Encerrada
Publisher	Sync	Conexão Encerrada	Conexão Encerrada

Conexão e desconexão - Subscriber com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Subscriber	Requester	Conexão Encerrada	Conexão Encerrada
Subscriber	Replyer	Conexão Encerrada	Conexão Encerrada
Subscriber	Ventilator	Conexão Encerrada	Conexão Encerrada
Subscriber	Worker	Conexão Encerrada	Conexão Encerrada
Subscriber	Sync	Conexão Encerrada	Conexão Encerrada

Conexão e desconexão - Ventilator com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Ventilator	Requester	Conexão Encerrada	Conexão Encerrada
Ventilator	Replyer	Conexão Encerrada	Conexão Encerrada
Ventilator	Publisher	Conexão Encerrada	Conexão Encerrada
Ventilator	Subscriber	Conexão Encerrada	Conexão Encerrada

Conexão e desconexão - Worker com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Worker	Requester	Conexão Encerrada	Conexão Encerrada
Worker	Replyer	Conexão Encerrada	Conexão Encerrada
Worker	Publisher	Conexão Encerrada	Conexão Encerrada
Worker	Subscriber	Conexão Encerrada	Conexão Encerrada

Conexão e desconexão - Sync com outros padrões			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Sync	Requester	Conexão Encerrada	Conexão Encerrada
Sync	Replyer	Conexão Encerrada	Conexão Encerrada
Sync	Publisher	Conexão Encerrada	Conexão Encerrada
Sync	Subscriber	Conexão Encerrada	Conexão Encerrada

Fonte: Autor.

6.2.3 Teste 3 - Conexão e desconexão de comportamentos incompatíveis no mesmo padrão

Devido a incompatibilidade de dois comportamentos no padrão Pipeline, também foram realizados testes pra averiguar se os dispositivos são desconectados nessas situações. Os resultados obtidos estiveram de acordo com o desejável, como apresentado na **Figura 24**.

Figura 24 – Resultados do teste 3, Tipos incompatíveis no mesmo padrão.

Conexão e desconexão - Comportamentos incompatíveis no mesmo padrão			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Ventilator	Sync	Conexão Encerrada	Conexão Encerrada

Fonte: Autor.

6.2.4 Teste 4 - Conexão e desconexão de comportamentos compatíveis

Os testes descritos nessa subseção foram realizados para averiguar se a conexão entre os dispositivos é mantida quando os tipos são compatíveis de conexão. Os resultados obtidos estiveram de acordo com o esperado para que os padrões funcionem corretamente. Os resultados estão descritos na **Figura 25**.

Figura 25 – Resultados do teste 4, Tipos compatíveis.

Conexão e desconexão - Tipos compatíveis			
Dispositivo 1	**Dispositivo 2**	**Resultado Obtido**	**Resultado Esperado**
Requester	Replyer	Conexão Mantida	Conexão Mantida
Publisher	Subscriber	Conexão Mantida	Conexão Mantida
Ventilator	Worker	Conexão Mantida	Conexão Mantida
Worker	Sync	Conexão Mantida	Conexão Mantida

Fonte: Autor.

6.2.5 Teste 5 - Envio e recebimento de dados

Os testes descritos aqui foram realizados com a finalidade de atestar que as funcionalidades de envio e recebimento de dados estão funcionando corretamente e como esperado. Nesse teste os resultados obtidos também estiveram de acordo com o esperado, como demonstrado na **Figura 26**.

Figura 26 – Resultados do teste 5, Envio e recebimento de dados.

Envio e recebimento de dados			
Origem	**Destino**	**Resultado do envio**	**Resultado do recebimento**
Publisher	Subscriber	OK	OK
Requester	Replyer	OK	OK
Replyer	Requester	OK	OK
Ventilator	Worker	OK	OK
Worker	Sync	OK	OK

Fonte: Autor.

6.2.6 Teste 6 - Subscrição de serviço

Os testes dessa subseção foram realizados para assegurar que a funcionalidade de subscrição de serviço esteja funcionando corretamente. Os resultados dos testes também apresentaram resultados desejáveis como pode ser visualizado na **Figura 27**.

Figura 27 – Resultados do teste 6, Subscrição de serviço.

Subscrição de serviço		
Origem	Destino	Resultado
Subscriber	Publisher	OK

Fonte: Autor.

6.2.7 Teste 7 - Confirmação de conclusão de processamento

Por fim, os testes dessa subseção foram realizados com a finalidade de assegurar o funcionamento da funcionalidade de comunicar a conclusão de processamento pelos comportamentos *Ventilator* e *Worker*. Os resultados obtidos dos testes estão de acordo com o esperado. Esses resultado estão descritos na **Figura 28**.

Figura 28 – Resultados do teste 7, Conclusão de processamento.

Comunicação de conclusão de processamento		
Origem	Destino	Resultado
Ventilator	Worker	OK
Worker	Sync	OK

Fonte: Autor.

6.3 Testes de processamento e armazenamento

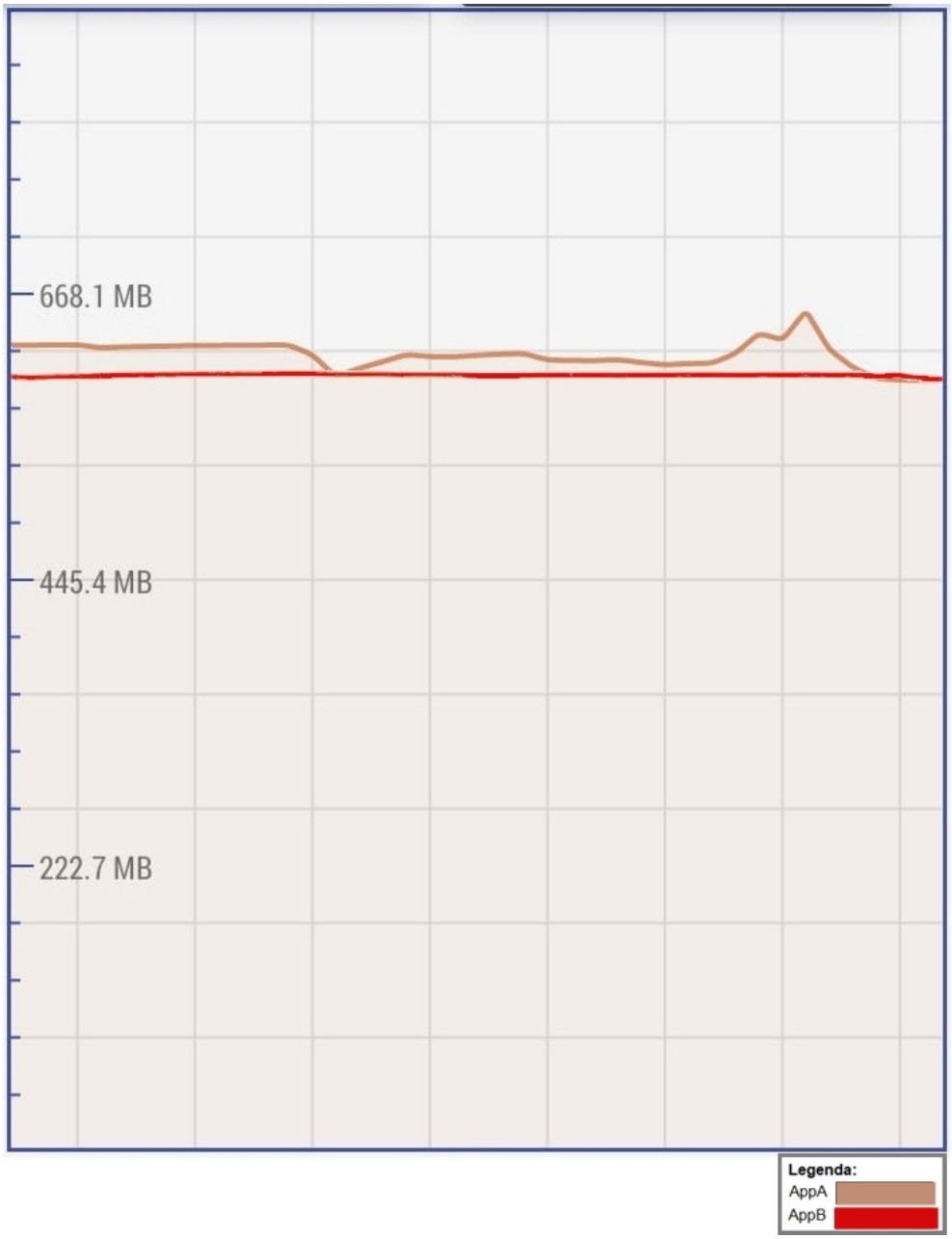
Nessa sessão serão apresentados testes comparativos de uso de processamento e memória entre uma aplicação "A"e uma aplicação "B". A aplicação "A"utilizará a Nearby Connections diretamente para realizar uma simples troca de mensagens, já a aplicação "B"fará uso da melhoria proposta para realizar a mesma ação. Os dados referentes à processamento e uso de memória serão apresentados antes do inicio de cada aplicação e durante o uso da aplicação. Os dados foram obtidos por meio do aplicativo Simple System Monitor. Cada unidade, representada

nos gráficos por quadrados, significa cinco segundos no eixo X do gráfico. O dispositivo utilizado possui 1Gb de memória RAM e 1.6GHz de frequência no processador.

6.3.1 Estado do dispositivo antes de iniciar as aplicações de teste

Como demonstrado pela **Figura 29**, os usos de memória RAM antes do inicio de cada aplicação de teste é bem semelhante, rondando em torno do mesmo patamar, além de serem bastante estáveis. Porém, a aplicação A, que não faz uso da melhoria proposta, apresentou um uso maior de memória.

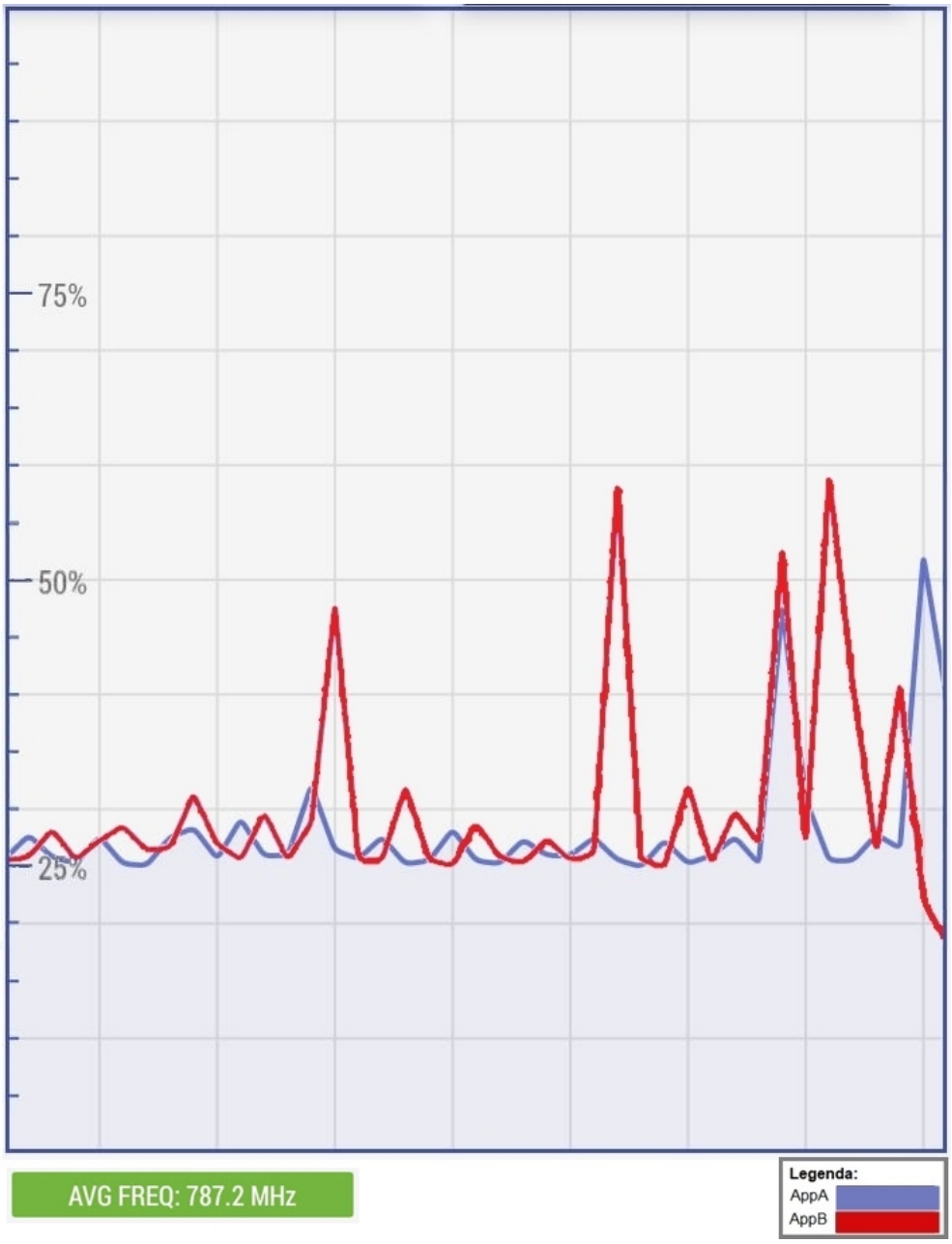
Figura 29 – Uso de memória RAM antes do inicio das aplicações de teste.



Fonte: Autor.

Essa mesma semelhança pode ser vista nos níveis de uso do processador, como demonstrado na **Figura 30**. O uso de processamento antes do funcionamento de ambas aplicações esteve em torno de vinte e cinco por cento, apresentando alguns picos de uso de processamento, que ocorrem ao alternar para o Simple System Monitor para realizar a captura dos dados por meio de capturas de telas. Em ambas as situações a frequência média de uso do processador foi de 787.2 MHz. A aplicação A acabou diferindo da aplicação B somente por apresentar uma quantidade menor de picos de uso de processamento.

Figura 30 – Uso do processador antes do inicio das aplicações de teste.

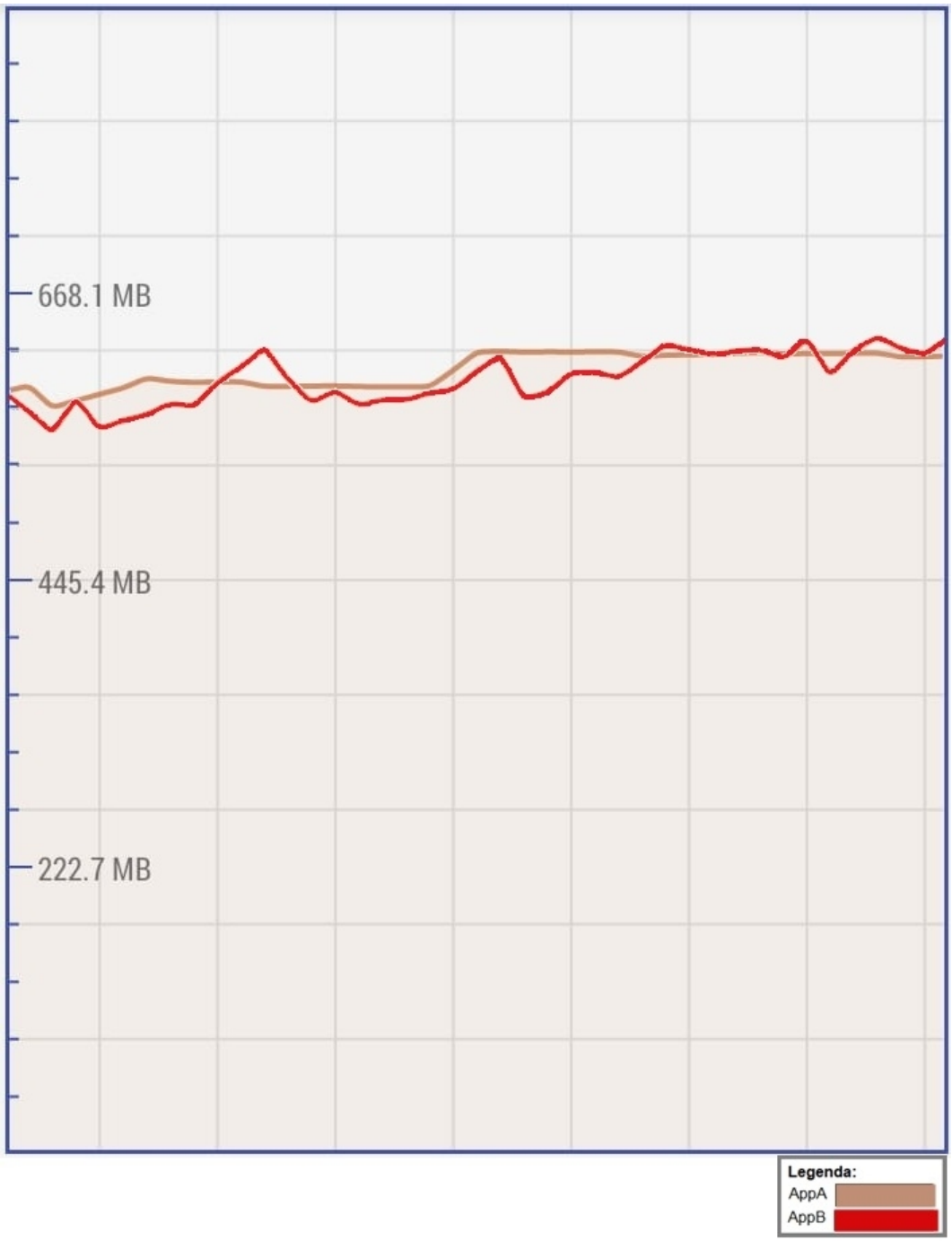


Fonte: Autor.

6.3.2 Estado do dispositivo durante o uso das aplicações de teste

Os usos de memória RAM apresentados na **Figura 31** não demonstraram diferença significativa quando comparados com os resultados obtidos na **Figura 29**. Entretanto, os resultados foram menos constantes, apresentando mais oscilações no uso de memória, principalmente na aplicação "B", que utiliza a melhoria proposta.

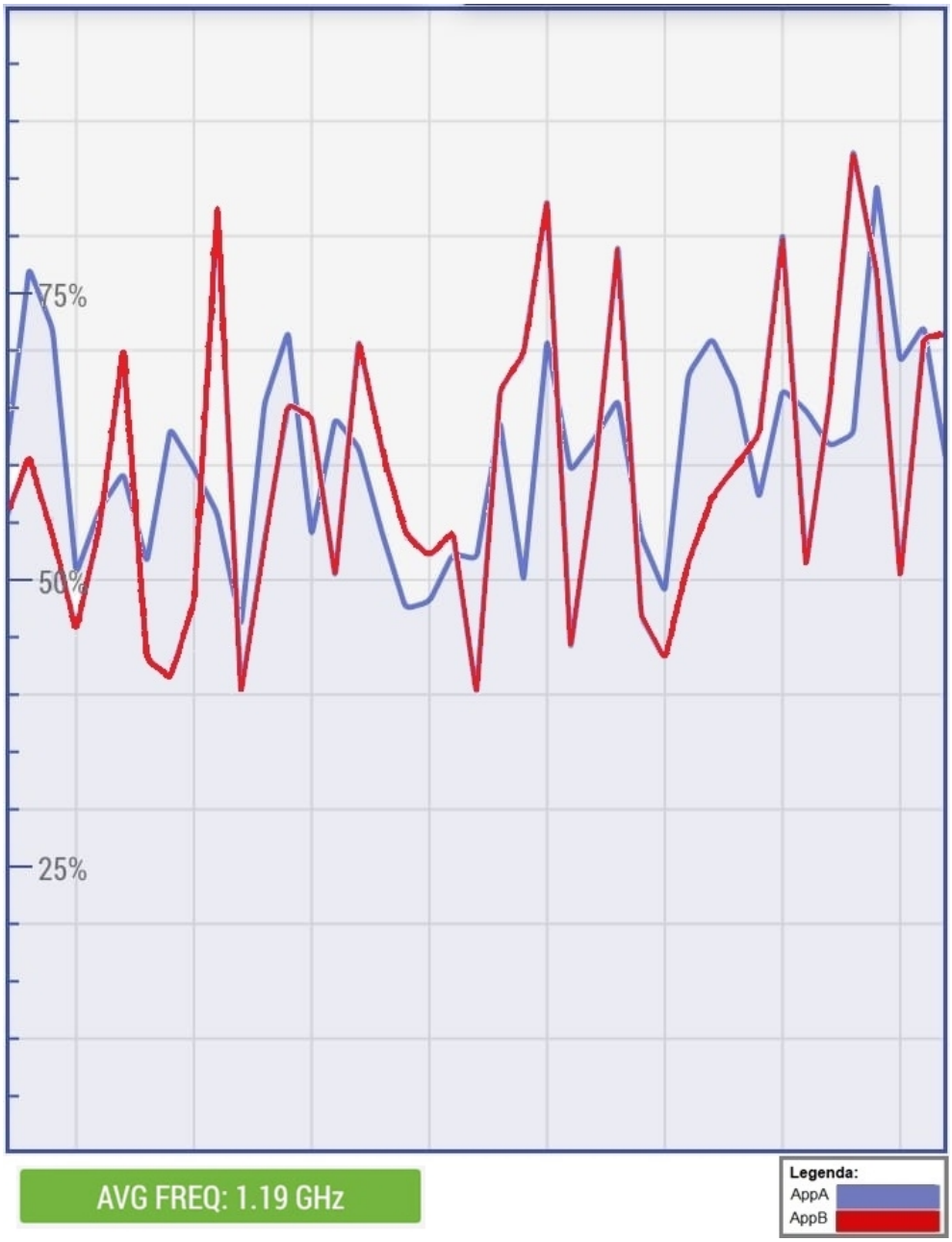
Figura 31 – Uso de memória RAM durante o uso das aplicações de teste.



Fonte: Autor.

O uso de processamento apresentado durante a execução de ambas as aplicações apresentou oscilações mais bruscas, sendo mais evidente na aplicação "B", onde os picos e as baixas de uso são maiores. Entretanto, a frequência média de uso do processador para ambas as aplicações foi de 1.19GHz.

Figura 32 – Uso do processador durante o uso das aplicações de teste.



Fonte: Autor.

6.4 Comparação de agilidade de desenvolvimento entre as aplicações A e B

As aplicações utilizadas para realizar a comparação de agilidade de desenvolvimento são as mesmas aplicações "A"e "B"utilizadas na seção anterior. Para medir a agilidade de desenvolver cada um das aplicações será utilizada a fórmula citada por Mikio Ikoma *et al.* (2009) em seu trabalho. Com isso, será utilizado o número de commits efetuados em cada aplicação para um período de um dia, que foi o tempo utilizado para desenvolver cada uma delas.

6.4.1 Agilidade de desenvolvimento da aplicação A

Devido ao fato da fórmula utilizada necessitar do número de entregas intermediárias como denominador da divisão, seria impossível realizar a medição para a aplicação "A", pois a mesma possui apenas dois commits, não possuindo commits intermediários dado que a mesma possui dois commits ao total. Para contornar essa limitação iremos supor uma entrega intermediária. Dessa modo o nível de agilidade para desenvolver a aplicação A resulta no valor 2, como demonstrado adiante na **Figura 33**.

Figura 33 – Agilidade de desenvolvimento da aplicação A.

```
E = V / U
V = 2
U = 1
E = 2 / 1 = 2
```

Fonte: Autor.

6.4.2 Agilidade de desenvolvimento da aplicação B

Para a aplicação B o calculo poderá ser feito diretamente já que a mesma já possui 3 commits originalmente, sendo um deles um commit intermediário. Desse modo, o nível de agilidade de desenvolvimento da aplicação B resulta em 3, sendo um pouco menos ágil que a aplicação A.

Figura 34 – Agilidade de desenvolvimento da aplicação B.

```
E = V / U
V = 3
U = 1
E = 3/1 = 3
```

Fonte: Autor.

7 CONCLUSÕES E TRABALHOS FUTUROS

Apesar das dificuldades apresentadas para realização dos testes propostos, principalmente pela dificuldade de emulação virtual do hardware de Bluetooth, os testes das regras introduzidas na melhoria apresentaram resultados satisfatórios, com todos os testes apresentado o resultado esperado para que os padrões de comunicação introduzidos funcionem corretamente. Os testes de comparação de desempenho também apresentaram bons resultados, comprovando diferenças pequenas no uso de processamento e memória quando comparado uma aplicação que faz uso direto da Nearby Connections e outra aplicação que faz uso da Nearby Connections para o mesmo proposito, porém através da melhoria proposta nesse trabalho. Apesar dos resultados anteriores apresentarem bons fundamentos, a melhoria proposta apresentou resultados menos expressivos, dada a métrica e o contexto escolhido, no quesito de agilidade de desenvolvimento, quando comparado com o uso direto da API. Quanto a melhoria semântica, pode-se assegurar um nível maior de qualidade advinda do uso desses padrões.

Algumas melhorias ainda podem ser feitas em trabalhos futuros para garantir mais qualidade e desempenho da melhoria proposta. A primeira delas, e mais importante para a manutenibilidade da melhoria, seria a realização de uma refatoração de código de modo a diminuir o acoplamento no código-fonte da melhoria. Desse modo, seria mais simples manter, melhorar e introduzir novos papéis. A segunda melhoria diz respeito a implantação de um sistema de tratamento de exceções mais robusto que possa informar de forma mais ampla e com mais qualidade quando uma exceção ocorrer. Por fim, sugere-se também para trabalhos futuros, a realização de testes em contextos mais complexos em projetos reais para comparação com os resultados obtidos nos testes efetuados nesse trabalho, além do uso de uma métrica para medir e comparar a facilidade semântica de aplicações que fazem uso da melhoria proposta e outras que não a utilizem.

REFERÊNCIAS

AKGUL, F. **ZeroMQ**. [S.l.]: Packt Publishing, 2013. 14 p. ISBN 9781782161042.

ANTONIOLI, D.; TIPPENHAUER, N. O.; RASMUSSEN, K. Nearby threats: Reversing, analyzing, and attacking google's' nearby connections' on android. 2019.

BENQ MOBILE GMBH CO OHG. **Ad Hoc Networking API Specification (JSR-259)**. [S.l.], 2006. Disponível em: <https://download.oracle.com/otndocs/jcp/adhoc_networking-0.1-edr-oth-JSpec>. Acesso em: 11 sep. 2019.

BRIJESH. **Nearby Connections API 2.0**. 2018. Disponível em: <https://androidkt.com/nearby-connections-api-2-0/>. Acesso em: 15 set. 2019.

GOOGLE. **Nearby Connections API Overview**. Google, 2017. Disponível em: <https://developers.google.com/nearby/connections/overview>. Acesso em: 30 ago. 2019.

GOOGLE. **Run apps on the Android Emulator : Android Developers**. Google, 2017. Disponível em: <https://developer.android.com/studio/run/emulator#limitations>. Acesso em: 20 abr. 2020.

GOOGLE. **Strategy Google APIs for Android Google Developers**. Google, 2017. Disponível em: <https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/Strategy>. Acesso em: 30 ago. 2019.

HINTJENS, P. **ZMQ - The Guide**. ØMQ Org, 2010. Disponível em: <http://zguide.zeromq.org/>. Acesso em: 25 ago. 2019.

HUI, P.; CHAINTREAU, A.; GASS, R.; SCOTT, J.; CROWCROFT, J.; DIOT, C. Pocket switched networking: Challenges, feasibility and implementation issues. In: SPRINGER. **Workshop on Autonomic Communication**. [S.l.], 2005. p. 1–12.

HUI, P.; CHAINTREAU, A.; SCOTT, J.; GASS, R.; CROWCROFT, J.; DIOT, C. Pocket switched networks and human mobility in conference environments. In: ACM. **Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking**. [S.l.], 2005. p. 244–251.

KURKOWSKI, S.; CAMP, T.; COLAGROSSO, M. Manet simulation studies: the incredibles. **ACM SIGMOBILE Mobile Computing and Communications Review**, ACM, v. 9, n. 4, p. 50–61, 2005.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top-down**. [S.l.]: Pearson Education do Brasil, 2013. 382–384 p. ISBN 9788543014432.

LIN, Y.-B.; CHLAMTAC, I.; SONS, J. W. . **2G1330 Mobile and Wireless Network Architectures**. 2001. 432 p.

Mikio Ikoma; Masayuki Ooshima; Takahiro Tanida; Michiko Oba; Sanshiro Sakai. Using a validation model to measure the agility of software development in a large software development organization. In: **2009 31st International Conference on Software Engineering - Companion Volume**. [S.l.: s.n.], 2009. p. 91–100.

MURTHY, C. S. R.; MANOJ, B. S. **Ad hoc wireless networks: Architectures and protocols, portable documents**. [S.l.]: Pearson education, 2004. 101 p. ISBN 0-13-147023-X.

TSENG, Y.-C.; LI, Y.-F.; CHANG, Y.-C. On route lifetime in multihop mobile ad hoc networks. **IEEE Transactions on Mobile Computing**, v. 2, n. 4, p. 366–376, Oct 2003.