Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul – UEMS

# *BLUECHATROOM*: UM APLICATIVO PARA CRIAÇÃO DE SALA VIRTUAL VIA *BLUETOOTH*

Lucas Ferrari de Sena e Borges

Prof. Dr. Evandro Cesar Bracht (Orientador)

Dourados - MS

2021

*BLUECHATROOM*: UM APLICATIVO PARA CRIAÇÃO DE SALA VIRTUAL VIA *BLUETOOTH*

Lucas Ferrari de Sena e Borges

Este exemplar corresponde à redação final da monografia da disciplina Projeto Final de Curso devidamente corrigida e defendida por Lucas Ferrari de Sena e Borges e aprovada pela Banca Examinadora, como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Dourados, 10 de dezembro de 2021.

Prof. Dr. Evandro Cesar Bracht(Orientador)

B732b Borges, Lucas Ferrari de Sena e
Bluechatroom: um aplicativo para criação de sala virtual via bluetooth / Lucas Ferrari de Sena e Borges. – Dourados, MS: UEMS, 2021.
40 p.

Monografia (Graduação) – Sistemas de Informação – Universidade Estadual de Mato Grosso do Sul, 2021.
Orientador: Prof. Dr. Evandro Cesar Bracht

1. Comunicação 2. Mobile 3. Bluetooth I. Bracht, Evandro Cesar II. Título

CDD 23. ed. - 006

Curso de Sistemas de Informação
Universidade Estadual de Mato Grosso do Sul – UEMS

# *BLUECHATROOM*: UM APLICATIVO PARA CRIAÇÃO DE SALA VIRTUAL VIA *BLUETOOTH*

Lucas Ferrari de Sena e Borges

Novembro de 2021

**Banca Examinadora:**

Prof. Dr. Evandro Cesar Bracht(Orientador)
Área de Computação - UEMS

Prof. Me. Delair Osvaldo Martinelli Júnior
Área de Computação - UEMS

Prof. Dr. Ricardo Luís Lachi
Área de Computação - UEMS

# Resumo

O presente trabalho descreve o desenvolvimento de um aplicativo que provê comunicação entre *smartphones* através de mensagens utilizando a tecnologia *bluetooth*, criando uma rede onde os aparelhos se comportam como nós. Esta rede pode possuir uma extensão física limitada apenas pelo alcance do sinal de rádio utilizado pelo *bluetooth* de cada dispositivo. Os dispositivos podem enviar mensagens ou as receber e repassar para os nós vizinhos, o objetivo é fazer com que a mensagem chegue a todos os dispositivos conectados à rede.

**Palavras-chave**: comunicação, *bluetooth*, *mensagem*, *chat*, *bate-papo*, *mobile*, *android*.

# Sumário

# Lista de ilustrações

# 1 Introdução

O *bluetooth* é uma tecnologia de comunicação sem fio existente desde 1994, que permite a transmissão de dados entre diversos tipos de dispositivos (KLEINSCHMIDT, 2004). Presente na maior parte dos *smartphones* mais recentes, o *bluetooth* foi desenvolvido para permitir a troca de dados a uma curta distância, mas possui características que permitem a construção de redes com um alcance maior. Uma de suas vantagens é a comunicação sem Internet.

De acordo com uma pesquisa publicada em 2019, estima-se que mais de 5 bilhões de pessoas tenham dispositivos móveis e mais da metade deles sejam *smartphones*. No Brasil, 60% dos adultos relataram possuir um *smartphone*, enquanto nos Estados Unidos o resultado foi 81% e na Coreia do Sul 95% (SILVER, 2019).

As redes sociais mais utilizadas (Facebook, Youtube e WhatsApp) (BELING, 2019), comumente empregadas na troca de informações entre dispositivos, são dependentes de conexão com a Internet, para tal o usuário precisa possuir um plano de dados móveis quando não estiver em uma rede Wi-Fi, o que pode ser considerado uma desvantagem quando se quer trocar informações entre dispositivos que não estão fisicamente distantes. Para suprir essa necessidade, é apropriado utilizar uma rede de comunicação que não necessite de Internet.

Visto que, atualmente o uso de *smartphones* é comum para grande parte da população mundial, a utilização de aplicativos faz parte do dia a dia e, a necessidade de trocar informações entre esses dispositivos é frequente. O desenvolvimento de um aplicativo com protocolos específicos para comunicação entre muitos dispositivos sem utilização de Internet é um trabalho relevante, que introduz possibilidades interessantes na área da comunicação sem fio.

## 1.1 Objetivos

Este projeto visa desenvolver um aplicativo de comunicação para dispositivos fisicamente próximos, limitado pelo alcance do *bluetooth*, realizado com base nos estudos sobre redes, topologias, e protocolos. A ideia principal é que o aplicativo crie uma rede em que cada dispositivo seja um nó em uma malha de conexões, na qual todos os aparelhos conectados tenham acesso às informações que estão sendo trocadas. Os dados que trafegarão na rede serão mensagens digitadas pelos usuários de cada dispositivo e também mensagens de controle ocultas para o usuário.

## 1.2 Justificativa

O *bluetooth*, diferente de outras tecnologias de redes sem fio, possibilita a conexão com vários dispositivos ao mesmo tempo. Mas esse recurso não é completamente aproveitado nas aplicações *bluetooth* mais comuns. Na realidade, apesar do *bluetooth* estar presente na grande maioria dos *smartphones*, passou a ser menos utilizado, à medida que houve o desenvolvimento de aplicativos que possibilitavam compartilhamento de arquivos via Wi-Fi mais rapidamente do que os que faziam o mesmo via *bluetooth*.

De certa forma, a ideia de que o *bluetooth* faz um uso muito alto da energia do dispositivo pode ter influenciado na desvalorização desse recurso no decorrer do tempo. No entanto, essa ideia está equivocada, como será explicitado na seção 3.4. Dessa forma, o uso do *bluetooth* no cotidiano passou a ser focado na conexão com dispositivos vestíveis (*wearables*) ou de mídia (como caixas de som). A partir dessa subutilização da capacidade de múltiplas conexões do *bluetooth* e o baixo uso popular dessa tecnologia, surge a ideia de se trabalhar em possibilidades ainda pouco exploradas, visto que o *bluetooth* possui características únicas nas suas capacidades de conexão que podem ser aproveitadas.

## 1.3 Metodologia

Neste trabalho foi realizada uma pesquisa descritiva e exploratória a respeito de redes de comunicação, da tecnologia *bluetooth*, protocolos existentes para redes *bluetooth* e análise da utilização de um protocolo específico para sustentar as necessidades do projeto, além do desenvolvimento do aplicativo. As fontes utilizadas são principalmente artigos, outros trabalhos científicos, e a documentação das tecnologias empregadas.

Para o desenvolvimento do aplicativo foi utilizada a *IDE Android Studio* juntamente com uma linguagem oficial de desenvolvimento para Android, o Java. O teste do funcionamento do aplicativo foi realizado com apenas três dispositivos, devido às restrições da pandemia do Coronavírus (COVID-19), o que impediu o contato com outros alunos, que participariam dos testes de campo utilizando o aplicativo. A partir dos testes realizados, foram identificadas limitações, falhas, melhorias, e também ideias de trabalhos futuros.

# 2 Redes

Em computação, uma rede é uma estrutura formada por um conjunto de dispositivos interconectados através de um sistema de comunicação com o objetivo de compartilharem informações entre si.

Existem diversos tipos de rede, sendo possível a classificação em categorias, baseando-se em critérios como dimensão ou área geográfica, topologia (ou ”a forma da rede”), meios físicos de suporte à rede, métodos de transferência, entre outros (TANENBAUM, 2003). Cada tipo de rede possui características específicas, sendo utilizadas em situações diferentes. Neste capítulo serão abordadas as características mais relevantes de alguns dos principais tipos de rede, incluindo vantagens e desvantagens para determinados casos, concluindo com a explicação do tipo de rede escolhido para o aplicativo.

## 2.1 Tipos de redes por dimensão

As redes podem ser classificadas de acordo com o seu alcance ou o espaço geográfico que ela é capaz de alcançar. Esse tipo de classificação será abordada nesta seção.

### 2.1.1 Redes LAN

As chamadas LANs, ou Redes Locais, interligam computadores presentes em um mesmo espaço físico. Amplamente usadas dentro de estações de trabalho, de universidades ou dentro da sua própria casa, sendo possível o compartilhamento de recursos e informações entre os dispositivos participantes (TANENBAUM, 2003).

### 2.1.2 Redes WLAN

WLANs são redes sem fio para transmissões em algumas dezenas de metros. WLAN é a sigla inglesa de *Wireless Local Area Network*, que em Português significa ”Rede Local Sem Fios”. É uma rede local que usa ondas de rádio para transmissão de dados e para conexão à Internet, sem necessidade de usar os tradicionais cabos para conectar dispositivos (CHLAMTAC MARCO CONTI, 2003).

### 2.1.3 Redes WAN

Uma rede WAN (*Wide Area Network*), também chamada de rede geograficamente distribuída ou rede de longa distância, abrange uma grande área, geralmente um ou mais países. (TANENBAUM, 2003).

### 2.1.4 Redes PAN

A PAN, do inglês *Personal Area Network*, é uma rede capaz de interligar aparelhos em uma área pessoal, com um alcance de até 10 metros. O objetivo deste tipo de rede é conectar os dispositivos pessoais de uma pessoa, como *notebook*, *smartphone*, *Smart TV* e os outros dispositivos do usuário. Esse tipo de rede também é chamada de WPAN quando utiliza conexão sem fios, geralmente utilizando *Bluetooth*, *Wi-Fi*, ou tecnologias semelhantes, enquanto uma PAN comum utiliza cabos, como USB e Ethernet. A PAN também pode ser utilizada para ligar determinados dispositivos a uma rede maior como a Internet (CHLAMTAC MARCO CONTI, 2003).

## 2.2 MANETs

As MANET ou Mobile Adhoc Network (Rede adhoc Móvel) são redes descentralizadas, onde não há um nó (dispositivo) principal ou essencial para o funcionamento da rede. Estas redes também são dinâmicas, o que significa que cada dispositivo pode se movimentar livremente, bem como se conectar e desconectar sem que existam problemas no funcionamento da rede. Devido às características desse tipo de rede, elas costumam utilizar topologia em malha, que é a mais adequada para esse tipo de topologia, pois sustenta de forma eficaz uma rede descentralizada e dinâmica. (CHLAMTAC MARCO CONTI, 2003)

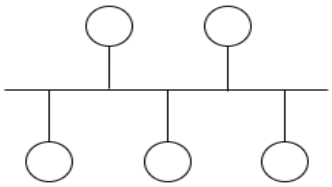
## 2.3 Topologias

Topologia em redes é o formato em que os dispositivos participantes de uma rede estão fisicamente organizados, sendo definida pela forma com que os dispositivos estão ligados entre si e quais responsabilidades eles possuem na rede. Existem diversas topologias que podem ser adotadas por uma rede, algumas das mais conhecidas são:

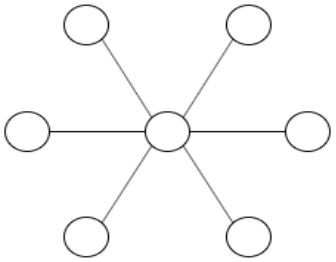
1. Barramento (ver **Figura 1a**).
2. Estrela (ver **Figura 1b**).
3. Anel (ver **Figura 2a**).
4. Malha (ver **Figura 2b**).

### 2.3.1 A topologia escolhida

Dentre as topologias listadas, a mais adequada para este aplicativo, é a MALHA, devido à possibilidade de descentralização. As redes em malha não possuem um nó central pelo qual todos os dados devem passar, nem mesmo um coordenador que defina as regras

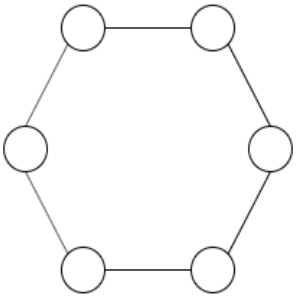


(a) Barramento.

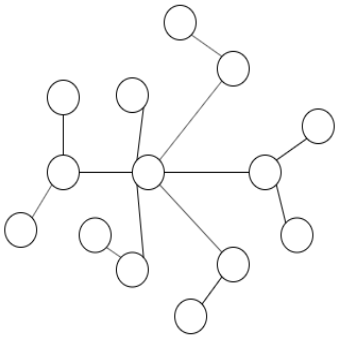


(b) Estrela.

Figura 1 – Barramento e Estrela.



(a) Anel.



(b) Malha.

Figura 2 – Anel e Malha.

da rede, tudo é definido em conjunto. Por isso, qualquer nó pode se desconectar da rede sem que a rede pare de funcionar por completo.

Porém, existem casos em que uma desconexão pode impactar grande parte da rede, como por exemplo caso um dispositivo atue como única ponte entre duas partes da rede, caso ele se desconecte, a rede se dividirá em duas, pois não haverá conexão entre todos os dispositivos. Por isso é importante que existam múltiplas conexões atuando como caminhos alternativos na rede, para garantir que não exista a dependência de um único nó.

# 3 Bluetooth

O *bluetooth* é uma tecnologia criada em 1994 para realizar o envio de dados entre dispositivos sem a utilização de fios, permitindo a comunicação entre quaisquer dispositivos que possuam o *hardware* do *bluetooth* instalado. (KLEINSCHMIDT, 2004)

O *bluetooth* realiza a comunicação por meio de ondas de rádio de curto alcance, utilizando frequências na faixa de 2.4Ghz a 2.485GHz. Essa faixa é chamada de *Industrial, Scientific and Medical* (ICM), e é de uso gratuito. (ARAUJO, 2012)

Para evitar as prováveis interferências, é usado o método de transmissão chamado *Frequency-hopping spread spectrum* (FHSS), onde os sistemas se comunicam em vários canais e realizam saltos aleatórios entre esses canais. (ARAUJO, 2012).

Neste capítulo serão abordados os tipos de redes que ele é capaz de formar, e suas vantagens no contexto do objetivo deste trabalho.

## 3.1 *Piconets*

O *bluetooth* possui a capacidade de se conectar a mais de um dispositivo simultaneamente, criando assim uma rede *Piconet* (BAKHSH HALABI HASBULLAH, 2010), como ilustrado na **Figura 3**. Nesse tipo de rede é definido um mestre o qual tem a responsabilidade de regular a transmissão de dados na rede e o sincronismo entre os dispositivos, os quais são chamados de escravos.

Em uma *Piconet*, o mestre pode estar conectado à uma quantidade limitada de escravos ao mesmo tempo, o que impede a criação de uma rede complexa usando apenas uma *Piconet*. No entanto, estabelecer conexões com mais dispositivos é possível através de outro tipo de rede *bluetooth*, a Scatternet.

## 3.2 *Scatternets*

Se um dispositivo *bluetooth* se conecta à duas *Piconets* simultaneamente e serve de ponte entre as *Piconets*, essa rede passa a ser classificada como uma *Scatternet* (SHARAFEDDINE IBRAHIM AL-KASSEM, 2011), ilustrada na **Figura 4**. Esse tipo de rede pode superar a limitação de quantidade de dispositivos que uma rede *bluetooth* pode possuir.

Diferente das *Piconets*, as *Scatternets* não são implementadas de forma automática no *bluetooth*, portanto a sua criação só pode ser feita via aplicação, ou seja, a estrutura de formação da Scatternet deve ser definida pelo programador.

Figura 3 – *Piconet*.

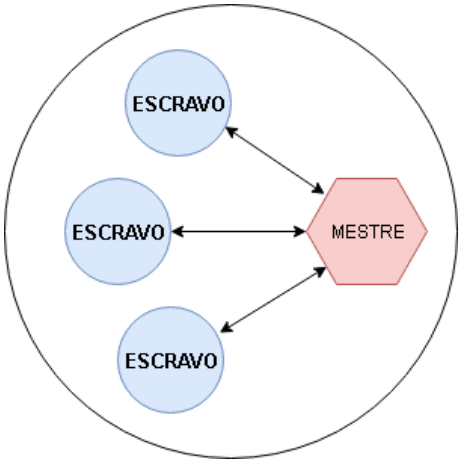
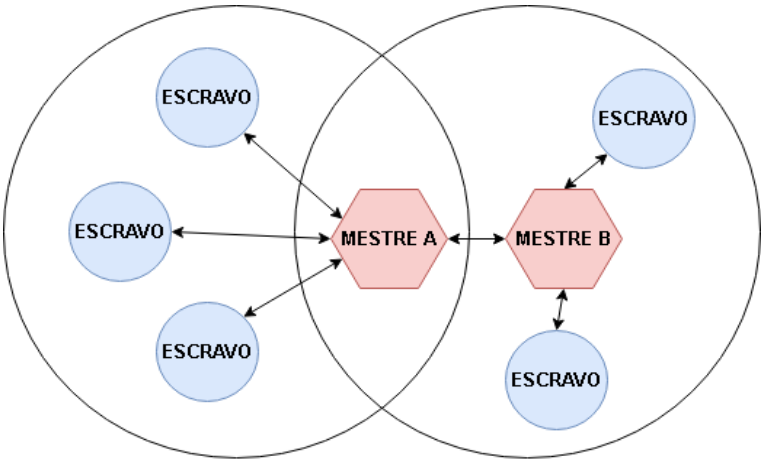



Figura 4 – *Scatternet*.




## 3.3 *A rede desenvolvida*

Como resultado do desenvolvimento, foi obtida uma rede WLAN com a topologia em malha gerada dinamicamente, semelhante a uma MANET. Ela é formada inicialmente por apenas um nó, que aceita receber conexões, onde um ou mais nós podem se conectar a este nó inicial, formando uma *piconet*. Caso um dispositivo se conecte a um dos nós secundários, formando uma ”linha direta” com três dispositivos, a rede se caracteriza como uma *scatternet*, por possuir um dispositivo atuando como ponte entre a *piconet* inicial e uma segunda *piconet*. A partir dessa situação, o tamanho da rede pode crescer de acordo com a quantidade de novos nós que se conectem a ela.

## 3.4 Consumo de bateria

Nas versões iniciais da tecnologia *bluetooth*, havia um gasto de bateria elevado, devido ao fato de que, quando ativado, o *bluetooth* buscava constantemente por dispositivos para conectar-se, o que acabava por consumir mais energia. No entanto, os desenvolvedores

do *bluetooth* conseguiram melhorar a tecnologia nessa questão, por isso, atualmente o gasto de bateria pelo *bluetooth* é muito menor.

Em 2010 a versão 4.0 do *bluetooth* trouxe consigo o BLE, do inglês *Bluetooth Low Energy*. Esta técnica permite diminuir os níveis de consumo de energia em aparelhos que não precisam transmitir grandes volumes de dados, podendo apresentar um gasto energético de apenas 10% em relação às versões anteriores do *bluetooth* (ARAUJO, 2012).

O BLE foi estudado, e constatou-se que ele não atende às necessidades do aplicativo, pois é voltado para comunicação dos *smartphones* com acessórios de baixíssima potência, como dispositivos vestíveis e sensores, sendo ideal para conexões que não precisam ser contínuas. Dessa forma, o *bluetooth* clássico foi a opção utilizada para a comunicação no projeto.

Contudo, o uso do *bluetooth* clássico não significa que o gasto de bateria será alto, pois isso é apenas um mito, causado pelo fato de que em versões antigas dessa tecnologia, o gasto energético era muito maior. Essa ideia se perpetuou através do tempo, mesmo que as diversas atualizações do *bluetooth* tenham diminuído drasticamente esse problema. (CANALTECH, 2021)

No que diz respeito aos módulos de sinalização e rede em um *smartphone*, o *bluetooth* é um dos componentes menos consomem energia, consumindo apenas 7% da energia total, um baixo gasto se comparado com outros módulos de comunicação, como a rede móvel (25%) e o Wi-Fi (25%). (PRAMANIK et al., 2019)

Dado o objetivo principal de criação do aplicativo, para uso em ambientes onde não é possível, ou não se deseja fazer o uso da comunicação por rede móvel ou Wi-Fi, a ideia é que essas tecnologias estejam desativadas no momento da utilização do aplicativo.

Por isso, ao atuar como nó da rede com o *bluetooth* ligado, o gasto de energia não se somaria ao das outras tecnologias. Ou seja, para casos onde não é necessário grande alcance de sinal, o bluetooth é uma opção mais econômica. Por isso, para os casos de uso específicos, a existência de uma alternativa é importante.

Dessa forma, qualquer receio em utilizar o aplicativo motivado apenas pelo gasto de energia do *bluetooth* não se justifica, visto que caso seja necessário enviar mensagens nas situações alvo deste projeto, seria necessário utilizar alguma tecnologia, e dentre as opções disponíveis o *bluetooth* é a mais econômica.

Quanto à real autonomia do dispositivo durante o uso deste aplicativo, os testes e análise necessária para se obter um resultado confiável exigem metodologias científicas rígidas e que demandam tempo e recursos que não estavam disponíveis durante o desenvolvimento do projeto. Por isso, só é possível apresentar os dados encontrados em pesquisa para dar uma noção básica de uso de energia da tecnologia *bluetooth*.

## 3.5 Capacidades da API Bluetooth no Android

Para que um aplicativo possa utilizar o *bluetooth* em dispositivos Android, o sistema oferece as chamadas APIs *Bluetooth* como abstração para funcionalidades do módulo de *hardware bluetooth*. Com a utilização das APIs para o *Bluetooth* Clássico, é possível realizar as seguintes atividades:

- Buscar outros dispositivos Bluetooth:

  A busca por outros dispositivos funciona de forma que ao ser iniciada, conforme encontra novos dispositivos visíveis, os adiciona na lista de encontrados para serem utilizados em uma futura solicitação de conexão.

- Parear-se com outros dispositivos:

  Pareamento é o procedimento em que um dispositivo solicita uma conexão pela primeira vez com outro. Nesse momento, caso o dispositivo receptor aceite a solicitação, acontece uma troca de informações e o armazenamento mútuo de (endereços, identificadores ou chaves de segurança), para que eles possam se identificar como ”confiáveis” e não ser necessária uma confirmação para que se estabeleça uma conexão futura. Um detalhe relevante: essas informações podem ser removidas a qualquer momento para revogar a permissão de conexão automática, então um novo pareamento deve ser realizado para que exista uma nova conexão.

- Verificar se existem dispositivos pareados:

  Durante o pareamento, são armazenados dados capazes de identificar o dispositivo pareado, e então esses dados (nome, endereço, etc), são inseridos na lista de dispositivos pareados. Esta lista pode ser exibida a qualquer momento no *smartphone*, desde que o *Bluetooth* esteja ligado.

- Estabelecer canais de comunicação:

  Os canais estabelecidos utilizam o protocolo RFCOMM (Protocolo de Comunicações de Rádio Frequência), um protocolo que emula portas seriais via ondas de rádio, utilizado como base para a conexão entre os dispositivos e possibilitando a troca de dados por esses canais. (SIG, 2012)

- Transferir dados entre dispositivos:

  Todo tipo de dado pode ser enviado via *bluetooth*, pois a base do *bluetooth* se comunica por meio de unidades de dado simples, com um fluxo de bytes, que são traduzidos após a recepção.

- Gerenciar as conexões:

O Gerenciamento de conexões é basicamente o controle sob quando e com qual dispositivo é realizada uma conexão. Podendo ser escolhido pelo usuário ou simplesmente pelo *software*.

# 4 Desenvolvimento do sistema

Para a criação da rede, foi desenvolvido um aplicativo para dispositivos *Android*, o que irá gerenciar as conexões entre os nós da rede, além de ser o responsável por transmitir e exibir as mensagens de texto em cada dispositivo.

Neste capítulo será abordado todo o desenvolvimento do aplicativo, incluindo a definição da política escolhida para a rede, conceitos da programação para *Android*, aparência do aplicativo, comunicação por *bluetooth*, e a utilização de classes e métodos necessários para a construção do *software*.

## 4.1 Requisitos de *software*

O aplicativo segue os seguintes requisitos funcionais de software:

- Buscar redes:

  Descrição: é possível procurar por redes nos arredores.

  Processo: mediante solicitação do usuário, o dispositivo busca por redes e informa ao usuário assim que as encontra.

- Solicitar conexão à rede:

  Descrição: qualquer dispositivo será uma rede por si só, e poderá se conectar a outras redes estabelecidas por outros dispositivos.

  Processo: mediante solicitação do usuário, um dispositivo fornece os seus dados de identificação e solicita acesso a um dispositivo que já faça parte da rede.

- Desconectar-se voluntariamente da rede:

  Descrição: o usuário tem a opção de se desconectar da rede a qualquer momento por meio de um botão.

  Processo: mediante solicitação do usuário, o aplicativo encerra todas as conexões ativas com outros dispositivos.

- Conceder acesso à rede:

  Descrição: um dispositivo tem a capacidade de permitir que outros dispositivos acessem a rede para receber e enviar mensagens.

  Processo: ao receber uma solicitação, o dispositivo concede acesso ao solicitante e o adiciona à lista de conectados para transmissão de novas mensagens.

- Envio de mensagens:

  Descrição: as mensagens são enviadas de um dispositivo para outro.

  Processo: O usuário digita uma mensagem e seu dispositivo envia para os dispositivos que estiverem diretamente conectados.

- Recebimento de mensagens.

  Descrição: As mensagens enviadas são recebidas por todos os dispositivos diretamente conectados.

  Processo: O dispositivo recebe uma mensagem e a processa de acordo com o protocolo de comunicação.

- Reencaminhamento de mensagens.

  Descrição: Ao receber uma mensagem, o dispositivo a repassa aos outros nós da rede.

  Processo: A mensagem recebida é repassada a todos os dispositivos diretamente conectados para que ela seja disseminada.

- Exibir os nós ativos na rede.

  Descrição: Um botão está disponível para que o usuário visualize uma lista com os nomes de todos os outros usuários online.

  Processo: Na rede são transmitidas mensagens para informar quando dispositivos conectam-se ou desconectam-se na rede, para que cada dispositivo possua uma lista local de todos os dispositivos online.

## 4.2 Modelo da Rede

Para o funcionamento da aplicação, é necessário que a rede seja dinâmica pois *smartphones* podem estar sempre em movimento. Para tolerar desconexões repentinas, bem como novas conexões constantes, a rede deve se adaptar conforme os dispositivos entram e saem da rede, sem que isso interfira no seu funcionamento adequado.

Assim, o ideal é que a rede seja descentralizada, ou seja, não exista um nó com papel especial ou central, todo dispositivo pode se desconectar a qualquer momento sem que isso afete a conexão de dispositivos que não estejam em condição de dependência dele.

Uma condição de dependência ocorre quando um dispositivo A só possui acesso à rede por meio da conexão com um dispositivo B, ou seja, se o dispositivo B se desconectar da rede, o dispositivo A deixa de receber as mensagens que estão trafegando. Contudo, o dispositivo A pode voltar à rede caso consiga conexão com outro dispositivo que esteja ligado na rede.

Para garantir que a rede possua as características citadas, será adotado o modelo de rede classificado como MANET. O *bluetooth* será responsável por gerenciar o *hardware* da conexão, e o aplicativo cuidará da definição do protocolo de compartilhamento da rede. Essa MANET terá como base a *Scatternet* criada pelo *bluetooth* juntamente com as regras estabelecidas via *software* para estabelecer a conexão com todos os dispositivos simultaneamente.

## 4.3 Desenvolvimento para Android

A plataforma escolhida para o sistema foi o sistema operacional Android, especificamente para *smartphones*, por sua popularidade e facilidade de desenvolvimento. Além do suporte nativo ao *bluetooth* na maioria dos dispositivos.

O Android Studio, o Ambiente de Desenvolvimento Integrado ou em inglês *Integrated Development Environment* (IDE) oficial da Google para desenvolvimento de aplicações *Android*, foi escolhido, tanto por ser uma ferramenta popular no mercado e com uma comunidade ativa, quanto por ser a ferramenta mais completa atualmente.

## 4.4 Comunicação via Bluetooth

Nesta seção será apresentado o procedimento de conexão com outros dispositivos utilizando *bluetooth*, desde a descoberta de dispositivos até o envio e recebimento de dados. Ao longo do texto serão inseridas imagens exemplificando o uso dos comandos mencionados.

### 4.4.1 Configuração do bluetooth

Para utilizar o *bluetooth* de um aparelho Android, é necessário configurar o *software* para tal, sendo um primeiro passo, a solicitação ao usuário das permissões de uso dos recursos. As permissões básicas para a utilização do *bluetooth* são:

- BLUETOOTH:

  Esta permissão engloba toda a utilização do módulo de *hardware bluetooth*. Qualquer *software* que utilize o *bluetooth* deve possuir essa permissão para funcionar plenamente.

- ACCESS_FINE_LOCATION:

  Esta permissão é utilizada quando se deseja determinar a localização mais exata possível, podendo utilizar dados móveis, Wi-Fi ou preferencialmente GPS. A partir

do Android 9 (API de nível 28), é possível substituir esta permissão pela ACCESS_COARSE_LOCATION, que determina a localização utilizando dados móveis ou Wi-Fi, porém com uma menor precisão.

Segundo a documentação oficial o motivo de ser necessário ativar a localização é que "Seu aplicativo precisa dessa permissão porque uma verificação *Bluetooth* pode ser usada para coletar informações sobre o local do usuário. Essa informação pode vir do próprio dispositivo do usuário, assim como de sensores *Bluetooth* em uso em locais como centros comerciais e de grande circulação."

- BLUETOOTH_ADMIN:

  Esta permissão é necessária para que novos dispositivos sejam encontrados. Mas além disso, essa permissão concede acesso a outros recursos, como configurações avançadas do *Bluetooth*, os quais não é recomendável utilizar na maioria dos aplicativos. Estes recursos são voltados para aplicativos com funcionalidades específicas de gerenciamento da energia do aparelho, que podem por exemplo, com a permissão do usuário, gerenciar de forma autônoma quando o *Bluetooth* fica ativo ou não. No caso do presente projeto, apenas os recursos básicos de busca de dispositivos serão utilizados.

A declaração das permissões no código do aplicativo é feita no arquivo AndroidManifest.xml (ver **Figura 5**). Esse arquivo deve existir em qualquer aplicativo.

Na área da computação, o arquivo de manifesto existe em diferentes ambientes de desenvolvimento e de forma geral, armazena metadados relevantes a respeito do *software*.

No caso de aplicativos para o sistema operacional Android, esse arquivo descreve informações essenciais sobre o aplicativo, que são utilizadas de diversas formas pelo ecossistema de aplicativos Android, como por exemplo:

- O nome do pacote do aplicativo
- Os componentes do aplicativo
- As permissões necessárias para o aplicativo
- Os recursos de *hardware* e *software* necessários

Além disso, existem muitas outras informações que podem ser declaradas no manifesto da aplicação.

Figura 5 – Declaração de permissões.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

### 4.4.2 Encontrando dispositivos

Após configurado, o próximo passo para realizar uma conexão via *bluetooth* é encontrar o dispositivo desejado. Para isso é necessário executar método `startDiscovery()` a partir de uma instância da classe `BluetoothAdapter` , a qual é responsável pelo gerenciamento do adaptador *bluetooth*, que é uma abstração para o *hardware* do *bluetooth*.

É importante que antes de iniciar a busca, seja feita uma checagem das permissões concedidas, pois a concessão é necessária para realizar a busca. Caso o *app* não possua as permissões, é necessário solicitar ao usuário.

É possível cancelar o descobrimento de dispositivos simplesmente executando o método `cancelDiscovery()` do objeto `bluetoothAdapter` , cancelando imediatamente a busca.

No método criado para realizar a busca, primeiro é feita a verificação de permissões, utilizando o método `checkPermissions()` . Depois é verificado se já existe uma busca em andamento, por meio do método `isDiscovering()` do `bluetoothAdapter` . Caso já exista uma busca em andamento, ela é cancelada, para que uma nova busca seja realizada (ver **Figura 6**).

Figura 6 – Encontrar dispositivos.

```
public void findDevices(View view){
    checkPermissions();

    if(bluetoothAdapter.isDiscovering()){
        bluetoothAdapter.cancelDiscovery();
    }
    bluetoothAdapter.startDiscovery();
}
```

Enquanto a busca de dispositivos está ocorrendo, sempre que há uma nova descoberta, é automaticamente emitido um evento de *broadcast*. Esse tipo de evento é ouvido por uma instância de `BroadcastReceiver` , que deve tratar os eventos de acordo com seus tipos.

O `broadcastReceiver` recebe o evento por meio do método `onReceive()` , que recebe como parâmetros o contexto, e um objeto da classe `Intent` , que contém os dados sobre o evento (ver **Figura 7**).

Ao receber o evento, o primeiro passo é verificar seu tipo, e ação a ser realizada, que pode ser obtida por uma chamada do método `getAction()` do `intent` , caso seja o tipo correto, o método prossegue normalmente. O tipo de evento emitido nesse caso será `BluetoothDevice.ACTION_FOUND` , constante incluída na classe `BluetoothDevice` (ver **Figura 7**).

O objetivo do método criado é armazenar as informações dos dispositivos assim que são encontrados, para que seja possível realizar conexões futuras. Dessa forma, os dispositivos, representados por objetos do tipo `BluetoothDevice` , são inseridos em uma lista de dispositivos encontrados. Esse objeto é obtido por meio do método `getParcelableExtra(extra)` do `intent` , passando como parâmetro um indicador de qual tipo de dado extra se deseja obter do `intent` . Nesse caso o valor desse parâmetro deve ser a constante `EXTRA_DEVICE` , contida em `BluetoothDevice` (ver **Figura 7**).

Figura 7 – *BroadcastReceiver* da busca.

```
private final BroadcastReceiver actionFoundReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            //Adiciona o dispositivo "device" na lista de encontrados.
        }
    }
};
```

### 4.4.3 Ativando a visibilidade

Para que um dispositivo seja encontrado em uma busca, é necessário que ele esteja visível (configuração que deve ser feita em cada dispositivo caso o usuário permita que seu dispositivo seja encontrado por outros), o que pode ser feito sem sair ou interromper o uso do aplicativo BlueChatRoom.

É importante ressaltar que ativar a visibilidade é uma configuração que implica na diminuição de privacidade e segurança, por isso é importante que a ação seja iniciada pelo usuário e ele permita que isso ocorra. Por isso o aplicativo não inicia esse processo de forma automática, ao ser executado.

Para que o usuário inicie o processo de ativação, é disponibilizada uma opção chamada “Tornar Visível”, exibida em um botão na tela inicial.

É possível solicitar a confirmação do usuário utilizando um recurso da API Android chamado `Intent` , que nesse caso irá exibir uma mensagem com opções de confirmar e

cancelar.

Para que o clique no botão acione o `Intent` , ele deve executar o método `startActivity(activity)` , onde o parâmetro "activity" é uma instância da classe `Intent` .

Em seguida, o sistema operacional exibe uma mensagem informando ao usuário que o aplicativo deseja ativar a visibilidade, oferecendo as opções de confirmar ou cancelar.

Para que o `Intent` seja criado corretamente, deve-se fornecer, na criação da instância a ação a ser realizada. Nesse caso a ação é `ACTION_REQUEST_DISCOVERABLE` , do `BluetoothAdapter` , utilizada para ativar a visibilidade do *bluetooth*.

A configuração de visibilidade por padrão possui um tempo limite de 120 segundos, e após esse tempo a visibilidade é desativada automaticamente. Mas caso o programador deseje, é possível utilizar o método `putExtra()` do `Intent` em conjunto com o valor `EXTRA_DISCOVERABLE_DURATION` do `BluetoothAdapter` , o que permite adicionar uma duração diferente, com limite de até uma hora (ver **Figura 8**).

Figura 8 – Método tornar visível.

```
public void tornarVisivel(View view){
    Intent discoverableIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
    discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, value: 300);
    startActivity(discoverableIntent);
}
```

Também é possível passar um valor zero como parâmetro para o tempo, de forma a manter o dispositivo detectável por tempo indeterminado. Mas isso não é recomendado, pois é uma configuração não segura.

### 4.4.4 Pareamento

Quando um dispositivo é encontrado, para que seja possível realizar uma conexão segura, é necessário realizar um procedimento chamado de pareamento ou emparelhamento, que consiste em realizar a troca de informações como nome, classe, endereço MAC (Media Access Control ou Controle de Acesso de Mídia) e chaves de segurança entre os dois dispositivos envolvidos, de forma que eles possam adicionar um ao outro em suas listas de dispositivos pareados.

A primeira parte do processo consiste em executar, em qualquer um dos dois dispositivos, o método `createBond()` , a partir da instância da classe `BluetoothDevice` obtida na busca de dispositivos (ver **Figura 9**). Após isso, o procedimento é feito de forma padrão pelo sistema Android, por isso o aplicativo não controla o que ocorre durante essa etapa.

Figura 9 – Criar pareamento.

```
if(vizinho.getBluetoothDevice().getBondState() == BluetoothDevice.BOND_NONE){
    vizinho.getBluetoothDevice().createBond();
}
```

De forma geral, o sistema apenas exibe uma notificação de solicitação de pareamento em cada um dos dispositivos, e então os usuários podem selecionar a opção de permitir, ou negar em caso de desacordo. Dessa forma, caso os dois lados aceitem, cada um adiciona o outro em sua lista de dispositivos confiáveis.

A partir desse momento, os dois dispositivos podem realizar conexões seguras a qualquer momento sem que seja necessário realizar um emparelhamento. Porém, o pareamento pode ser revertido simplesmente removendo o par em qualquer um dos dois dispositivos, fazendo com que seja necessário realizar um novo pareamento quando uma nova conexão for desejada.

### 4.4.5 Estabelecimento de conexão

Quando dois dispositivos estão pareados, é possível criar uma conexão. Para isso, antes, deve ser executado o método `createRfcommSocketToServiceRecord()` da classe `BluetoothDevice` obtida na busca de dispositivos. Esse método cria um objeto instância de `BluetoothSocket` , o qual contém todos os dados da conexão, e os métodos necessários para realizar a comunicação entre eles e ficará responsável pelo gerenciamento da conexão com este dispositivo.

A partir de então é possível executar o pedido de conexão, por meio do método `connect()` da instância de `BluetoothSocket` . Após isso o dispositivo fica aguardando uma resposta, se a resposta for positiva, a conexão é estabelecida, caso contrário, uma exceção é lançada no aplicativo.

Quando a conexão é estabelecida com sucesso, o *socket* criado deve ser persistido na memória do aplicativo para que seja possível gerenciar a conexão a qualquer momento, enviando ou recebendo dados.

A conexão permanece ativa enquanto os dispositivos estiverem dentro da área de alcance do *bluetooth* um do outro, mas pode ser encerrada utilizando um método do `BluetoothSocket` chamado `close()` .

### 4.4.6 Envio de dados

As mensagens podem ser enviadas por meio de pequenos pacotes de bytes. Para o envio dos pacotes, o *bluetooth* permite que se utilize o método `write()` da classe

`OutputStream` , fornecida pela instância de `BluetoothSocket` gerente da conexão.

Primeiro se obtém uma instância de `OutputStream` como resposta ao executar o método `getOutputStream()` do objeto de `BluetoothSocket` . A partir de então, deve-se executar o comando `outputStream.write(array)` sendo ”array” um arranjo/vetor de bytes contendo os dados a transmitir.

Ao chamar esse método, os bytes do array são enviados para o dispositivo *bluetooth* associado ao `BluetoothSocket` em questão.

### 4.4.7 Recebimento de dados

Para obter os dados enviados por um dispositivo, o receptor deve obter os bytes a partir do método `read()` da classe `InputStream` contida no `BluetoothSocket` associado ao remetente.

Para que seja possível identificar quando algum dado foi enviado, o receptor deve executar um laço de repetição no qual é feita a verificação recorrente dos dados contidos no *buffer* (ou *array*) de recebimento de dados. Caso exista algum conteúdo, o mesmo deve ser tratado, e então feita a limpeza do *array* (*buffer*).

## 4.5 Protocolo de comunicação utilizado

Após estabelecida uma conexão via *bluetooth*, a transmissão de dados deve ser solicitada a nível de aplicação, ou seja, os comandos para enviar e receber em uma conexão devem ser feitos por meio do código do aplicativo. Portanto, foi criado, um padrão para o tratamento das informações trocadas entre os dispositivos.

De acordo com Falbriard (FALBRIARD, 2002), “os protocolos utilizados em redes de comunicação definem conjuntos de regras que coordenam e asseguram o transporte das informações úteis entre dois ou mais dispositivos”. Dessa forma, as regras criadas durante o desenvolvimento desta aplicação, podem ser chamadas simplesmente de protocolo. Nesta seção será descrito o protocolo utilizado na comunicação entre os nós da rede.

### 4.5.1 Mensagens

A comunicação é realizada por meio do envio simples de bytes, que são em seguida convertidos em texto (*string*). O mínimo de bytes enviados em uma mensagem válida é de 69, enquanto o máximo é 984.

Para que toda a informação necessária seja transmitida em apenas um envio, cada mensagem é dividida em 4 partes, sendo cada uma responsável por carregar um dado diferente (ver **Figura 10**). São elas:

- TYPE - Contém o tipo da mensagem, informando o objetivo da comunicação, para que a informação seja processada corretamente (1 byte);
- UUID - Contém o identificador único da mensagem, criado uma vez e permanece inalterado durante o tráfego da mensagem na rede (36 bytes);
- USER - Contém o nome do usuário que inicialmente gerou a mensagem, também chamado de proprietário ou remetente (1 a 17 bytes);
- TEXT - Contém o texto da mensagem, que de forma geral, é o conteúdo a ser exibido ou tratado, dependendo do contexto (de 1 a 900 bytes).

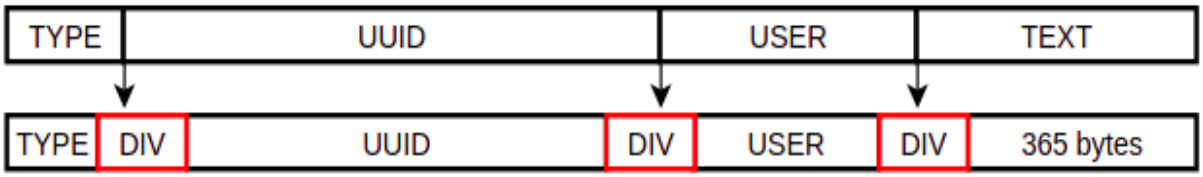
Figura 10 – Partes da mensagem.

| TYPE | UUID | USER | TEXT |
|---|---|---|---|
| 1 | eb708b5d-8dec-4196-98b0-d5fd286ac6f7 | Lucas Ferrari | Bom dia Grupo!! |
| 1 byte | 36 bytes | 1 a 17 bytes | 1 a 900 bytes |

### 4.5.2 A divisão da mensagem

Para que seja possível identificar onde é o início e o fim de cada parte da mensagem, a divisão é feita utilizando um pequeno conjunto de três caracteres especiais, com 9 bytes no total, que marcam a localização onde a *string* da mensagem completa deve ser "cortada" para que se obtenha cada informação separadamente. Os campos que dividem a mensagem são inseridos entre os campos principais (ver **Figura 11**).

Figura 11 – Divisão da mensagem.




### 4.5.3 TYPE - Tipos de mensagem

É um campo com tamanho fixo de 1 byte, utilizado para indicar como uma mensagem deve ser tratada, podendo ser de controle da rede ou a comunicação que deve ser exibida ao usuário. Os tipos de mensagem criados são:

- CHAT_ORIGIN - Informa que uma mensagem acaba de ser gerada, sendo o remetente o proprietário dela. Pode-se entendê-la como a mensagem original;

- CHAT_SHARE, - Informa que uma mensagem está sendo repassada, não pertencendo ao dispositivo que a está enviando. Pode-se entendê-la como uma cópia da mensagem original;
- INFO_ARRIVAL - Informa que um dispositivo acaba de se conectar à rede;
- INFO_FAREWELL - Informa que um dispositivo foi desconectado da rede, seja intencionalmente ou não;
- ERROR - Utilizada quando um pedido de conexão à rede é negado. O participante da rede que bloquear a conexão deve enviar uma mensagem desse tipo, contendo o motivo do erro;
- NONE - Utilizado quando uma mensagem é inválida por algum motivo. Caso necessário, o detalhamento é realizado no conteúdo da mensagem.

Para que cada tipo corresponda a um valor de apenas um byte, os tipos são convertidos em valores inteiros de 0 a 5, o que também possibilita que novos tipos sejam adicionados até o máximo de 10 (0 a 9) sem que o espaço ocupado na mensagem final se altere (ver **Figura 12**).

Figura 12 – Valor de cada tipo de mensagem.

| TIPO | VALOR |
|---|---|
| CHAT_ORIGIN | 0 |
| CHAT_SHARE | 1 |
| INFO_ARRIVAL | 2 |
| INFO_FAREWELL | 3 |
| NONE | 4 |

### 4.5.4 UUID - Identificadores de mensagens

UUID, do inglês *universally unique identifier*, traduzido em português como identificador único universal, é um código com tamanho fixo de 128 bits (16 bytes) usado para identificar informações em sistemas de computação. Esse tamanho é razoavelmente pequeno comparado com outras alternativas, o que, entre outras características, possibilita que o UUID seja uma boa alternativa para ordenação, armazenamento, e outros usos de *software* em geral, podendo ser utilizado desde a identificação de objetos de vida útil muito curta, até objetos persistentes de duração longa ou indefinida, sendo muito vantajosa para uma aplicação distribuída. (LEACH; MEALLING; SALZ, 2005)

No projeto, os UUIDs são utilizados para identificar cada mensagem enviada na rede, e nesse caso pode surgir a dúvida sobre a possibilidade de duas mensagens possuírem um mesmo UUID, causando conflito e um possível erro na aplicação, principalmente caso existam muitas mensagens sendo geradas ao mesmo tempo na rede. Mas é justamente esse o problema que o UUID se propõe a resolver, dado que o uso de identificadores básicos como por exemplo números inteiros é impossibilitado pela alta probabilidade de duplicidade.

Existem 5 versões do UUID, sendo que cada uma delas possui um algoritmo diferente para gerar o UUID. A versão utilizada neste projeto é a 4ª, que na linguagem Java é gerada a partir do método `randomUUID()` da classe `UUID` . Essa chamada gera uma instância de `UUID` , que pode ser convertida em *string* utilizando o método `toString()` . (BAELDUNG, 2021)

De acordo com Ludi Rehak, sobre a possibilidade de colisões de UUIDs, "Uma amostra de $3,26 * 10^{16}$ UUIDs tem 99,99% de chance de não ter duplicatas. E a geração de tantos UUIDs, por exemplo a uma taxa de um por segundo levaria um bilhão de anos. Portanto, embora os UUIDs não sejam verdadeiramente únicos, eles são únicos o suficiente para fins práticos, levando em consideração as limitações naturais da expectativa de vida humana e a separação dos sistemas." (REHAK, 2017)

Levando em consideração a singularidade padrão dos UUIDs na versão 4, e as características do presente projeto, uma duplicidade capaz de gerar um conflito nesta aplicação é virtualmente impossível de ocorrer, pois o UUID de uma mensagem só fica armazenado em cada dispositivo por apenas 15 segundos, para a checagem de possível duplicidade de recebimento.

### 4.5.5 USER - Proprietários de mensagens

Um usuário é identificado pelo seu nome de usuário na rede, que pode ser definido pelo proprietário do aparelho. O nome de usuário na rede deve ser único, não sendo permitida a entrada de um nó em uma rede que já possua um nó com um nome de usuário idêntico.

Os nomes de usuários também são utilizados para que um dispositivo exiba, quando solicitado, uma lista de todos os outros usuários atualmente online na rede.

O campo usuário possui de 1 a 17 bytes, limitando assim o tamanho mínimo e máximo do nome dos usuários na rede.

### 4.5.6 TEXT - Conteúdo de mensagens

O texto da mensagem é um campo de tamanho variável, com no mínimo 1 byte e no máximo 365 bytes. Essa parte traz o conteúdo, que deverá ser tratado pelo receptor de

acordo com o tipo da mensagem.

Caso a mensagem seja do tipo CHAT_ORIGIN ou CHAT_SHARE, o conteúdo é simplesmente utilizado na exibição para o usuário que recebeu a mensagem, pois se trata do texto digitado pelo remetente.

Caso a mensagem seja do tipo INFO_ARRIVAL, o conteúdo é interpretado como o nome do usuário que acaba de ingressar na rede, sendo utilizado em todos os nós receptores para adicionar o novo nó às suas listas de dispositivos online.

Caso a mensagem seja do tipo INFO_FAREWELL, o conteúdo é interpretado como o nome do usuário que foi desconectado, dessa forma, todos os nós podem removê-lo de suas listas de dispositivos online ao receberem a mensagem.

Caso a mensagem seja do tipo ERROR, o conteúdo é a mensagem de erro descrevendo o problema em questão.

### 4.5.7 Funcionamento geral do protocolo

Dadas as definições básicas do protocolo, agora será explicado como ocorre o fluxo dos dados na rede levando em conta os tipos de mensagem e os dados contidos em cada uma delas.

Quando um usuário conectado em um grupo digita o texto e em seguida clica no botão de enviá-lo, o seu dispositivo/*smartphone* gera uma mensagem contendo todos os dados anteriormente citados, tipo, UUID, usuário e texto. O UUID dessa mensagem é então armazenado em uma lista, onde esse dado ficará armazenado por 15 segundos. A mensagem criada é então enviada para cada um dos dispositivos conectados diretamente a esse nó.

Cada um dos nós que receberem uma mensagem, deve, primeiramente, verificar se eles já a receberam anteriormente, utilizando sua lista local com os UUIDs de mensagens já tratadas. Caso o UUID da mensagem que acaba de chegar já esteja armazenado em sua lista de UUIDs recentes, o receptor simplesmente ignora a mensagem, pois uma mensagem idêntica já foi tratada anteriormente. Caso o UUID não esteja presente na lista, o receptor deve adicioná-lo, exibir a mensagem, e distribuí-la para os próximos nós com os quais ele estiver conectado.

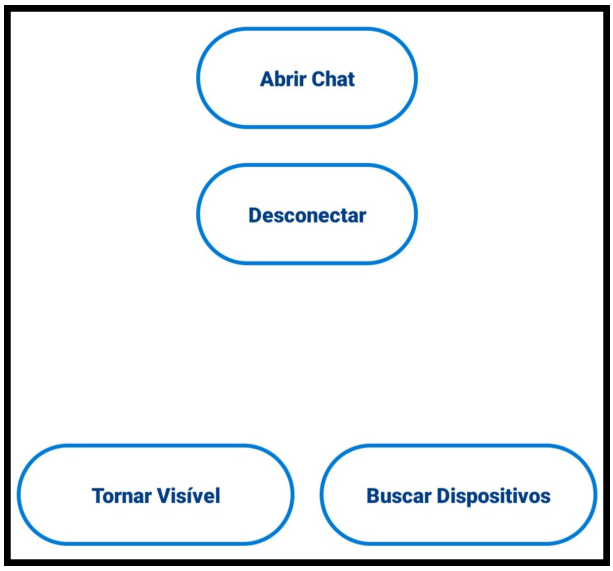
O protocolo não faz tratamento para garantir o sincronismo das mensagens, por isso elas são exibidas e reencaminhadas independentemente de data e hora de envio, sendo exibidas por ordem de chegada. Além disso, não existem mensagens de confirmação de recebimento, nem há garantia de que todos os participantes receberão uma mensagem, o que se deve principalmente à descentralização e dinamicidade da rede.

# 5 Utilização do aplicativo

A interface do usuário é o meio pelo qual o usuário realiza todas as ações no aplicativo, por isso é uma parte muito importante do aplicativo, e as informações devem ser claras para auxiliar o usuário durante a utilização. Nesta seção serão apresentadas as telas com as quais o usuário vai interagir, além das funcionalidades incluídas em cada uma delas.

Ao abrir o aplicativo, é visualizada a tela inicial com alguns botões (ver **Figura 13**). Caso o *bluetooth* esteja desligado, uma mensagem é automaticamente exibida para solicitar a ativação (ver **Figura 14**).

Figura 13 – Tela inicial.




Clicando no botão “Tornar Visível”, é exibida a mensagem de confirmação para a ativação da visibilidade (ver **Figura 15**).

Clicando em “Abrir *chat*”, é exibido o *chat* vazio, pois não há nenhum outro dispositivo conectado (ver **Figura 16**). O mesmo pode ser verificado clicando no ícone para visualização de participantes, no canto superior direito da tela, que exibe uma mensagem informando que o *chat* está vazio (ver **Figura 17**).

Clicando em buscar dispositivos, a busca é iniciada, e caso um ou mais dispositivos encontrados possuam o identificador de rede em seu nome, o aplicativo automaticamente informa que uma rede foi encontrada, e pergunta ao usuário se ele deseja se conectar (ver **Figura 18**).

Quando a conexão é bem sucedida, o *chat* é aberto automaticamente, e já é possível enviar mensagens de texto na sala de bate-papo. Durante a conversa, o nome do usuário

Figura 14 – Ativação do Bluetooth.

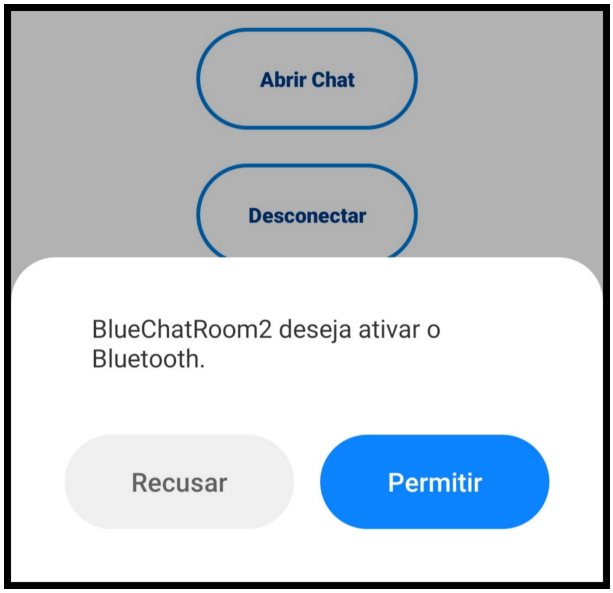
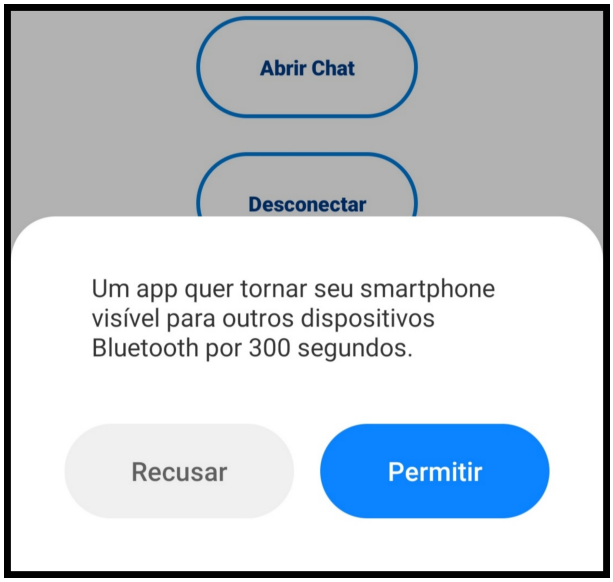



Figura 15 – Tornar visível.




proprietário de cada mensagem é exibido sob o balão de texto (ver **Figura 19**).

Quando existe um ou mais dispositivos conectados na rede, o ícone de visualização de participantes exibe uma lista com todos aqueles que estão conectados no momento (ver **Figura 20**).

É possível sair da conversa em grupo a qualquer momento usando o método padrão do *smartphone* para retornar, seja por meio do botão voltar ou por ação equivalente, o que pode variar de acordo com cada dispositivo. Essa ação realiza o retorno para a tela inicial do aplicativo.

Retornar para a tela inicial não desconecta o dispositivo da rede, apenas deixa de exibir o *chat* para o usuário. Sendo necessário apertar o botão “Desconectar” para que o

Figura 16 – Chat Vazio.


Figura 17 – Lista de pessoas online vazia.

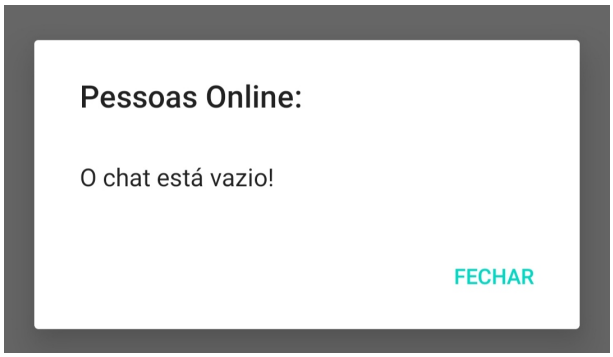
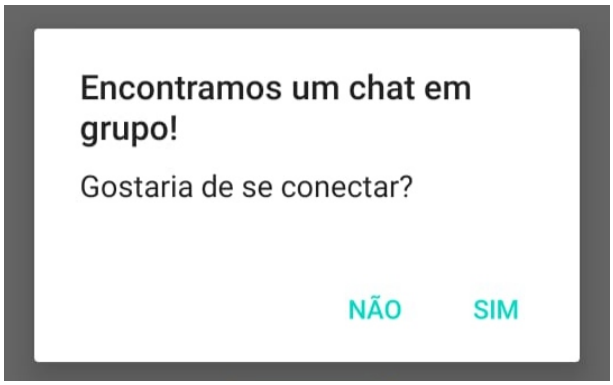



Figura 18 – Chat encontrado.




dispositivo se desconecte e deixe de ser um nó da rede.

Quando um dispositivo se desconecta da rede, uma mensagem flutuante temporária é exibida para todos os usuários informando qual usuário se desconectou. Essa notificação fica visível por alguns segundos e então desaparece (ver **Figura 21**).

Figura 19 – Bate-papo.

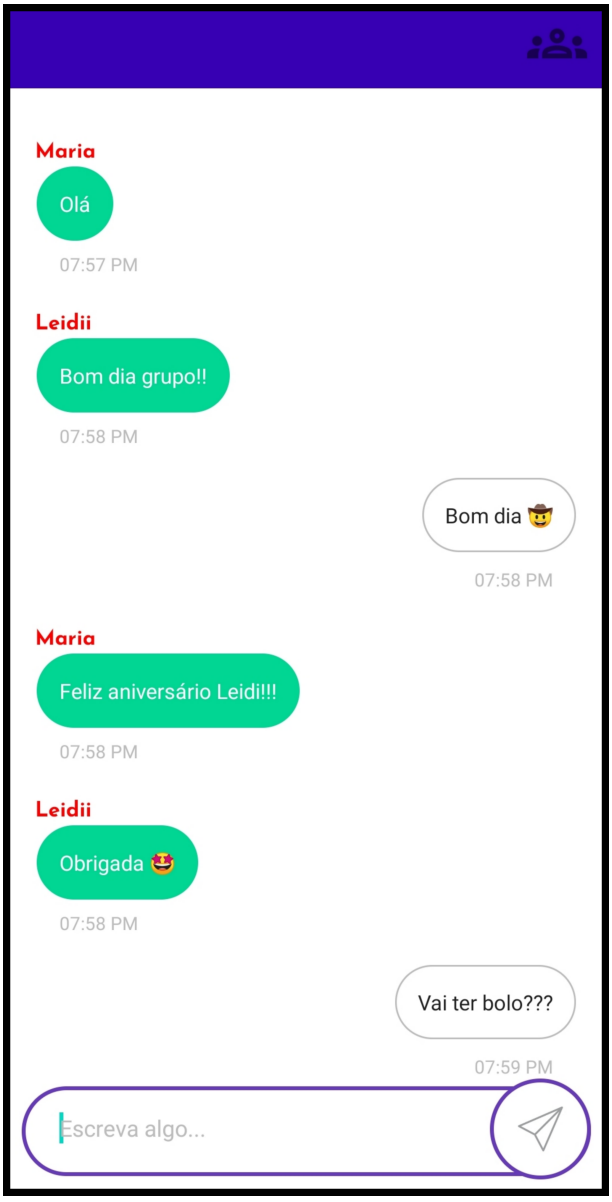



Figura 20 – Pessoas online.

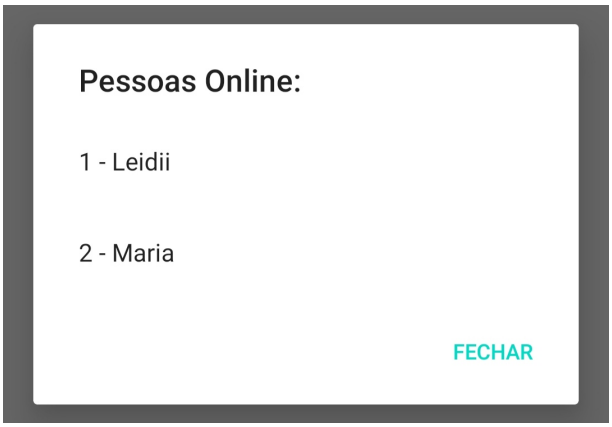


Figura 21 – Saída de usuário.

# 6 Conclusão

Como resultado do projeto, foi criado um aplicativo capaz de gerar uma rede com diversos participantes sem a necessidade de conexão com a Internet, apenas utilizando o *bluetooth*.

Por meio dessa rede são trocadas mensagens de texto escritas pelos usuários, e essas mensagens são exibidas para todos os participantes da rede. Dado que a comunicação por meios digitais é uma parte relevante do dia a dia das pessoas, o aplicativo se encaixa nesse contexto para suprir essa demanda onde a conexão com a internet é limitada ou inexistente, seja por falta de infraestrutura ou pelo custo que os usuários precisam ter para utilizarem planos de Internet.

Devido às restrições da pandemia do Coronavírus COVID-19, seguindo as recomendações da OMS de distanciamento social, não foi possível realizar um teste amplo com diversos dispositivos para criar uma rede complexa. Apenas testes com até três dispositivos foram realizados.

Apesar da restrição na quantidade de dispositivos utilizados nos testes, foi possível observar que o aplicativo funciona de acordo com o esperado.

## 6.1 Trabalhos futuros

Algumas sugestões para trabalhos futuros de melhorias no aplicativo e ideias relacionadas a esse tipo de rede são:

### 6.1.1 Melhorias no aplicativo

- Enviar mensagens como objeto:

  No aplicativo, a comunicação entre os dispositivos é baseada no envio de uma *string*, que é convertida em bytes para o envio e convertida novamente em *string* ao ser recebida.

  Esse processo pode ser realizado utilizando um objeto ao invés de uma *string*, de forma que os dados fiquem melhor estruturados, e não seja necessária uma classe para realizar a conversão dos dados, como ocorre utilizando *strings*.

- Permitir envio de outros tipos de conteúdo:

  No sistema desenvolvido, apenas mensagens de texto podem ser trocadas na rede, enquanto nos aplicativos mensageiros mais comuns, essa limitação não existe, são

trocados arquivos de todos os tipos, como imagens, vídeos e gifs. Esse tipo de arquivo seria uma ótima adição de funcionalidade ao aplicativo do BlueChatRoom.

- Mensagens diretas através da rede:

  A aplicação criada proporciona o envio das mensagens para um grupo, onde todos os participantes podem ver as mensagens, enviadas, como ocorre normalmente em outros aplicativos mensageiros. Mas a ideia da rede pode ser estendida de forma que os participantes possam se comunicar de forma privada, sendo que os nós intermediários apenas retransmitem as mensagens até seu destino, sem revelar seu conteúdo. Essa funcionalidade implicaria em uma grande adição de código no aplicativo, e seria necessário pensar em uma forma de criptografia, para que não seja possível que uma versão maliciosa do aplicativo revele as mensagens privadas.

- Melhoria na descentralização:

  Na versão atual do aplicativo, um dispositivo apenas faz a solicitação de conexão com vários dispositivos em dois casos:

  1 - Quando vários nós de rede são encontrados de uma vez só ao realizar a busca, pois o aplicativo solicita conexão com todos eles.

  2 - Quando o usuário sai da tela de bate-papo, realiza uma nova busca, encontra algum novo nó, e conecta-se a ele.

  Portanto, caso não ocorra nenhum desses casos, a rede torna-se mais centralizada do que o esperado no planejamento do projeto.

  Uma solução para essa centralização, seria exibir automaticamente para o usuário uma sugestão de buscar outros nós para se conectar, sem ter que sair do *chat* para isso.

  Outra solução seria realizar essa conexão de forma automática, sem a necessidade de ação do usuário, levando em consideração questões das permissões necessárias para isso e as boas práticas no desenvolvimento de aplicativos.

### 6.1.2 Ideia relacionada

Como uma rede móvel descentralizada utilizando *bluetooth*, existem possibilidades a serem exploradas. Uma dessas ideias será apresentada a seguir.

- Compartilhamento de recursos:

  Os dispositivos na rede podem listar os recursos de *hardware* ou *software* que possuem, como sensores, processadores, dados úteis, memória temporária ou programas de tratamento de dados, e então disponibilizá-los aos outros participantes.

Esse tipo de interação entre dispositivos é viável pois já existem aplicativos que fornecem funcionalidades desse tipo utilizando a câmera e microfone de dois dispositivos, por exemplo. A ideia seria realizar esse tipo de comunicação em uma rede mais ampla e com nós intermediários, sendo importante analisar os possíveis limitadores de comunicação, como a capacidade de transferência de dados do bluetooth.

# Referências


ARAUJO, P. d. V. André Silveira de. Bluetooth Low Energy. 2012. Disponível em: <https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2012_2/bluetooth/index.htm>. Acesso em: 23 nov. 2019. Citado 2 vezes nas páginas 13 e 15.

BAELDUNG. Guide to UUID in Java. 2021. Disponível em: <https://www.baeldung.com/java-uuid>. Acesso em: 14 out. 2021. Citado na página 29.

BAKHSH HALABI HASBULLAH, S. T. S. T. Dynamic congestion control through backup relay in bluetooth scatternet. Elsevier, 2010. Citado na página 13.

BELING, F. As 10 maiores redes sociais. 2019. Disponível em: <https://www.oficinadanet.com.br/post/16064-quais-sao-as-dez-maiores-redes-sociais>. Acesso em: 23 nov. 2019. Citado na página 8.

CANALTECH. Deixar o BLUETOOTH LIGADO faz mal pro celular e GASTA MAIS BATERIA? 2021. Disponível em: <https://www.youtube.com/watch?v=3eWUMREbzys>. Acesso em: 14 out. 2021. Citado na página 15.

CHLAMTAC MARCO CONTI, J. J.-N. L. I. Mobile ad hoc networking: imperatives and challenges. Elsevier, 2003. Citado 2 vezes nas páginas 10 e 11.

FALBRIARD, C. Protocolos e Aplicações para Redes de Computadores. [S.l.]: Editora Érica, 2002. 63 p. Citado na página 26.

KLEINSCHMIDT, J. H. Redes bluetooth : modelagem, desempenho e aplicações. Dissertação (diploma thesis) — Universidade Católica do Paraná, 2004. Citado 2 vezes nas páginas 8 e 13.

LEACH, P.; MEALLING, M.; SALZ, R. Rfc 4122: A universally unique identifier (uuid) urn namespace. 2005. Disponível em: <http://www.ietf.org/rfc/rfc4122.txt>. Citado na página 28.

PRAMANIK, P. K. D. et al. Power consumption analysis, measurement, management, and issues: A state-of-the-art review of smartphone battery and energy usage. IEEE Access, v. 7, p. 182113–182172, 2019. Citado na página 15.

REHAK, L. Are UUIDs really unique? 2017. Disponível em: <https://towardsdatascience.com/are-uuids-really-unique-57eb80fc2a87>. Acesso em: 14 out. 2021. Citado na página 29.

SHARAFEDDINE IBRAHIM AL-KASSEM, Z. D. S. A scatternet formation algorithm for bluetooth networks with a non-uniform distribution of devices. Elsevier, 2011. Citado na página 13.

SIG, B. RFCOMM WITH TS 07.10. 2012. Disponível em: <https://www.bluetooth.com/specifications/specs/rfcomm-1-2/>. Acesso em: 7 dez. 2021. Citado na página 16.

SILVER, L. Smartphone Ownership Is Growing Rapidly Around the World, but Not Always Equally. 2019. Disponível em: <https://www.pewresearch.org/global/2019/02/05/smartphone-ownership-is-growing-rapidly-around-the-world-but-not-always-equally/>. Acesso em: 23 nov. 2019. Citado na página 8.

TANENBAUM, A. S. Redes de computadores - 4ª Ed. [S.l.]: Editora Campus (Elsevier), 2003. Citado na página 10.