UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE SISTEMAS DE INFORMAÇÃO

MARCOS AIRES BORGES

BTMesh

Framework para a criação de redes mesh sobre Bluetooth LE

Canoas
2018

MARCOS AIRES BORGES

BTMesh

Framework para a criação de redes mesh sobre Bluetooth LE

Artigo apresentado como requisito parcial para obtenção do título de Bacharel em Sistemas de Informação, pelo Curso de Sistemas de Informação da Universidade do Vale do Rio dos Sinos - UNISINOS

Orientador: Prof. Ms. Ernesto Lindstaedt

Canoas

2018

BTMesh: Framework para a criação de redes mesh sobre Bluetooth LE

Marcos Aires Borges*
Ernesto Lindstaedt**

Resumo: Atualmente a comunicação entre dispositivos móveis acontece predominantemente através da Internet, que é dependente da infraestrutura de empresas de telefonia e comunicação. Se tais sistemas se tornarem indisponíveis, seja por falha eletrônica ou por saturação, seus usuários se encontrarão incapazes de trocar informações entre si. Como solução para este problema, o presente trabalho tem por objetivo a pesquisa e desenvolvimento de um framework móvel para a criação de redes mesh atuando sobre conexões Bluetooth Low Energy (BLE). É proposta a criação de uma malha dinâmica de comunicação local entre os dispositivos envolvidos, mesmo que estes não estejam conectados à Internet. Para tal, foi necessário pesquisar o funcionamento e limitações do padrão BLE, arquiteturas de rede e programação reativa, a fim de projetar um protocolo que seja performático em termos de uso de banda de dados e consumo de energia. Os resultados obtidos nos experimentos realizados demonstram viabilidade do framework proposto como agente de intercomunicação de dispositivos móveis utilizando tecnologia BLE, mesmo estando em um estágio inicial de desenvolvimento.

Palavras-chave: Bluetooth Low Energy. BLE. Programação Reativa. Redes Mesh.

1 INTRODUÇÃO

Com a evolução dos dispositivos móveis e a grande demanda dos usuários por permanecerem sempre conectados uns aos outros, cada vez mais estes aparelhos estão dependentes de infraestruturas tradicionais de rede para se comunicar, mas nem todos os locais possuem cobertura 3G/4G ou Wi-Fi - ou possuem, mas estas entram em estado de saturação, como no caso de eventos esportivos.

A ideia é que seja possível criar uma malha dinâmica de comunicação local entre os dispositivos envolvidos, mesmo que estes não estejam conectados à Internet.

Para tal, foi desenvolvido um framework utilizando tecnologia Bluetooth de baixo consumo de energia, por se tratar de um padrão aberto e que permite a

* Marcos Aires Borges, marcos.xraymob@gmail.com. Sistemas de Informação – UNISINOS – São Leopoldo/RS.
** Ernesto Lindstaedt, linds@unisinos.br. Orientador, Prof. Ms., UNISINOS – São Leopoldo/RS.

portabilidade do protocolo criado para várias plataformas e sistemas operacionais móveis.

1.2 Objetivos

Esta seção apresenta os propósitos do estudo e do projeto de desenvolvimento do BTMesh através do objetivo geral e objetivos específicos.

1.2.1 Objetivo Geral

Projetar e desenvolver um framework móvel, para a plataforma iOS, que através de sua utilização possibilite a criação e uso de uma rede mesh atuando sobre conexões Bluetooth Low Energy.

1.2.1 Objetivos específicos

a) Investigar o suporte da linguagem Swift[1] para as funcionalidades a serem implementadas;

b) Pesquisar frameworks que possibilitem o desenvolvimento reativo para a plataforma iOS;

c) Pesquisar arquiteturas de desenvolvimento móvel: MVC, MVVM, Viper (Orlov, 2015);

d) Realizar estudo prático quanto às limitações do padrão BLE na plataforma iOS, bem como testes de performance;

e) Estudar os modelos de arquitetura mesh existentes;

f) Pesquisar uma possível arquitetura para o protocolo de comunicação;

g) Desenvolver e implementar o framework;

h) Avaliar o framework.

1.2.1 Justificativa

Nos tempos atuais, verifica-se cada vez mais a vontade e a necessidade de comunicação por parte dos usuários de dispositivos móveis. Mas esta comunicação

[1] Linguagem de programação open source: https://swift.org

depende da infraestrutura da Internet para ocorrer, além de cobertura de sinal por parte das operadoras de telefonia móvel.

Mas o que acontece se a Internet se tornar inoperante? Ou se o sistema de dados de uma companhia telefônica não estiver disponível, no caso de uma catástrofe natural? Pensando nestes cenários, deseja-se verificar a viabilidade do desenvolvimento de um framework de software capaz de gerar uma malha de comunicação descentralizada que não dependa de nenhuma infraestrutura externa.

Existem soluções comercias para a criação de redes mesh em aplicativos móveis, como o MeshKit SDK (MeshKit), da empresa OpenGarden. Mas tais soluções além de pagas (preço não divulgado), não disponibilizam seu código-fonte de forma aberta.

O diferencial do BTMesh será justamente o fato que ele poderá ser melhorado e utilizado livremente pela comunidade, bem como portado para outros sistemas operacionais, como Android e Windows Phone por exemplo.

1.2.2 Organização do artigo

No capítulo 2 é explanado sobre o referencial teórico que embasou o presente trabalho. Já no capítulo 3 são elencados dois trabalhos relacionados ao tema do roteamento mesh sobre Bluetooth Low Energy. O capítulo 4 trata da metodologia utilizada para o levantamento de dados necessários para direcionar o desenvolvimento do framework e do seu desenvolvimento em si. O capítulo 5 trata da análise dos dados obtida, bem como apresenta o aplicativo de demonstração utilizado para testar o framework BTMesh. Por fim, o capítulo 6 apresenta as conclusões encontradas e também trata sobre itens de desenvolvimento futuro para a complementação do framework BTMesh.

2 REFERENCIAL TEÓRICO

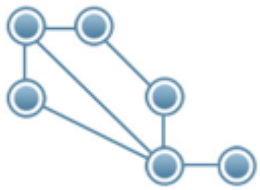
Serão introduzidos a seguir os principais conceitos estudados para a elaboração deste trabalho.

2.1 Topologia de rede do tipo malha (mesh)

Segundo Bandeira (2013), “A topologia descreve como os equipamentos estão interligados, ou seja, a forma física como eles estão interligados. A topologia tem a ver como os dados fluem na rede. [...] “. Existem vários tipos de topologias, com sua características e limitações. Este trabalho tem como um dos seus objetivos entender o funcionamento da topologia em malha (mesh).

A topologia mesh introduz o conceito de rota. Diferentemente de outras topologias, as mensagens enviadas na rede mesh podem seguir por vários caminhos distintos entre origem e destino. Muitas redes WAN, inclusive a própria Internet, utilizam roteamento mesh. Uma representação gráfica desta topologia pode ser vista na Figura 1:

Figura 1 - Topologia Mesh



Fonte: Certiology (2017)

Segundo Bandeira (2013), “[...] como temos vários links, se um deles falhar, podemos redirecionar os dados por outro caminho”. Isto se torna particularmente interessante quando pensamos em dispositivos que podem estar fisicamente em movimento, como aparelhos celulares. Neste caso o alcance da rede depende da disposição física dos envolvidos; duas redes distintas podem se interconectar se houver apenas um nó comum às duas.

2.2 O protocolo Bluetooth

Segundo Ray (2015), Bluetooth[2] é um padrão de comunicação sem fio para pequenas distâncias, operando na faixa entre 2400-2483.5 MHz, com 79 canais de 1MHz de largura de banda disponíveis. Esta tecnologia é normalmente utilizada para

[2] Bluetooth SIG (Special Interest Group). Entidade responsável pela supervisão do desenvolvimento do padrão Bluetooth: https://www.bluetooth.com

a conexão de computadores/smartphones e equipamentos acessórios, como por exemplo dispositivos de áudio e de entrada de dados (teclado e mouse).

2.2.1 Bluetooth Low Energy

Bluetooth Low Energy, ou BLE, é a versão 4.0 do padrão Bluetooth, cujo foco foi a redução do consumo de energia nos dispositivos móveis. BLE se difere por permanecer em modo de repouso e apenas realizar conexão quando solicitado, em rajadas menores que as versões anteriores (Ray, 2015).

2.2.1 RSSI (*Received Signal Strength Indication*)

A leitura RSSI indica a potência do sinal recebido em uma transmissão de rádio, no caso Bluetooth. O padrão de valores para esta leitura depende de cada fabricante (Gao, 2015).

No caso dos dispositivos iOS, observa-se que os valores RSSI se encontram entre 0 e -127 numa escala em decibéis, sendo os valores mais próximos de 0 os que indicam maior potência do sinal.

Como se trata de transmissão de sinais de rádio, a leitura RSSI pode variar consideravelmente, pois sofre influência do meio onde os dispositivos estão fisicamente dispostos. Neste caso, segundo Gao (2015), o melhor seria considerar a Moda (medida de tendência central de um conjunto de valores, que indica aquele que ocorre com mais frequência) dos valores, para eliminar o ruído nos dados.

2.3 Programação reativa

Neste projeto foram utilizados recursos de programação reativa, por se tratar de um framework de comunicação, onde é necessário observar sequências de dados e estado de conexões.

A programação reativa é baseada no padrão de desenvolvimento “Observer” (Silva, 2009), mas organizada de forma estruturada; o que a torna mais segura do ponto de vista de legibilidade de código e manutenção (Pillet, 2017). Ela é especialmente útil no BTMesh pois é possível “amarrar” diversos controles, como monitoramento de tempo de execução de comandos e decisões quanto ao fluxo de

dados, com o estado das conexões; algo que seria mais complexo e trabalhoso de ser executado utilizando-se exclusivamente programação imperativa.

2.4 A arquitetura da Plataforma iOS

2.4.1 Cocoa Touch

Cocoa Touch é o grupo de frameworks usado como base de desenvolvimento para a plataforma iOS. Ele inclui os frameworks Foundation e UIKit, entre outros.

Como dito em Apple (Outubro de 2015),

> The term “Cocoa” has been used to refer generically to any class or object that is based on the Objective-C runtime and inherits from the root class, NSObject. The terms “Cocoa” or “Cocoa Touch” are also used when referring to application development using any programmatic interface of the respective platforms.

O framework Foundation implementa a classe raiz, NSObject, que define o comportamento base dos objetos. Ele também possui os tipos primitivos (números e strings, por exemplo) e coleções (arrays e dicionários).

O framework UIKit é utilizado para a criação da interface visual da aplicação. Ele possui classes para lidar com eventos, desenhar na tela, processar imagens entre outros. Também estão incluídos aqui os elementos visuais de interface: botões, sliders, campos de input de texto, etc (Apple, Outubro de 2015).

2.4.2 Linguagem de programação Swift

A linguagem escolhida para o desenvolvimento do projeto foi a linguagem Swift, uma das duas linguagens oficiais de desenvolvimento iOS, por se tratar de uma linguagem moderna e bastante poderosa. Swift incorpora aspectos de programação funcional, além de ser uma linguagem bastante segura e fortemente tipada.

Segundo Mattias (2016), Swift é uma linguagem bastante expressiva, de fácil leitura e excelente para grandes projetos colaborativos. Estas características foram fundamentais para que a empresa Apple, criadora da linguagem Swift, anunciasse publicamente seu lançamento em 2014, como substituta da já desgastada linguagem Objective-C para o desenvolvimento de aplicações na plataforma iOS.

Em dezembro de 2015 a linguagem se tornou ainda mais relevante quando seu código-fonte foi disponibilizado de forma aberta. Isto fez com que a comunidade de desenvolvedores passasse a contribuir com a evolução da linguagem, bem como portá-la para diferentes plataformas (Apple, Dezembro de 2015).

2.4.3 A biblioteca RXSwift

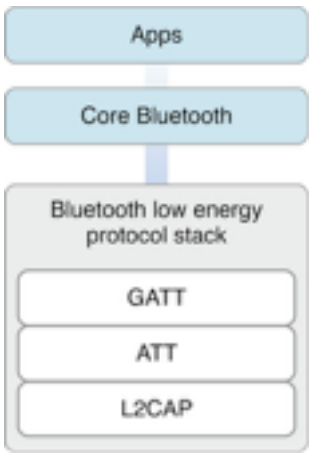
RXSwift é uma biblioteca para código assíncrono baseado em eventos, que se utiliza de sequências observáveis e operadores funcionais. Ela simplifica o desenvolvimento de código assíncrono pois proporciona que o código Swift reaja às atualizações na camada de dados, e processe estas de forma sequencial e isolada. Segundo Pillet (2017),

> RxSwift finds the sweet spot between traditionally imperative Cocoa code and purist functional code. It allows you to react to events by using immutable code definitions to process asynchronously pieces of input in a deterministic, composable way.

2.4.4 O framework Core Bluetooth

O framework Core Bluetooth fornecido pela Apple, cuja estrutura básica pode ser observada na Figura 2, permite o desenvolvimento de aplicações iOS utilizando-se da tecnologia Bluetooth Low Energy. Este framework abstrai alguns aspectos técnicos do funcionamento interno das conexões Bluetooth 4.0, permitindo ao desenvolvedor focar no uso prático de tal tecnologia (Apple, Setembro de 2013).

Figura 2 - Core Bluetooth Stack



Fonte: Apple (Setembro de 2013)

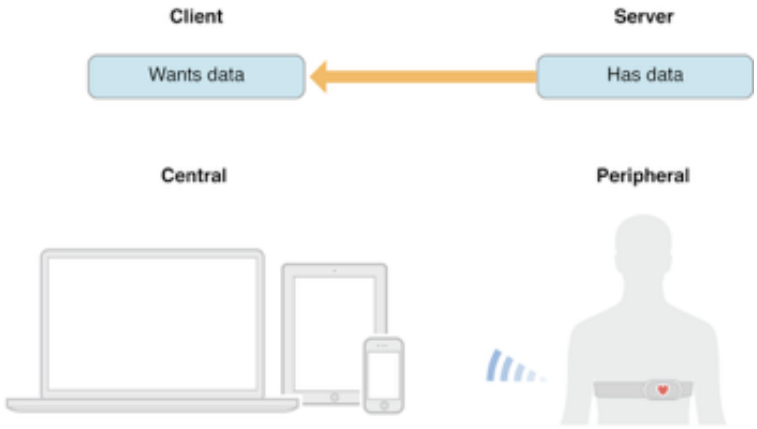
2.4.4.1 Papéis central e periférico na comunicação BT

Segundo BLE (2015), a comunicação no padrão Bluetooth LE é constituída por dois papéis principais: central e periférico.

O periférico é o elemento que fornece informação: em uma abordagem cliente-servidor, seria quem possui a informação que os outros dispositivos necessitam.

O central se utiliza da informação fornecida pelo periférico para realizar suas tarefas, como pode ser visto a seguir:

Figura 3 - Dispositivos Central e Periférico



Fonte: Apple (Setembro de 2013)

Para que o periférico seja visto pelo central, ele envia pacotes de formatação[3] (*advertising*), indicando sua existência e disposição de transmitir dados. Este pacote possui um tamanho reduzido, dispondo de informações básicas como o nome do dispositivo e tipo de dados que será fornecido. O elemento central faz *scanning*, ou seja, fica “escutando" os pacotes de *advertising* BLE para decidir se possui interesse em receber dados de algum periférico.

Figura 4 - *Advertising* e descoberta de dispositivos

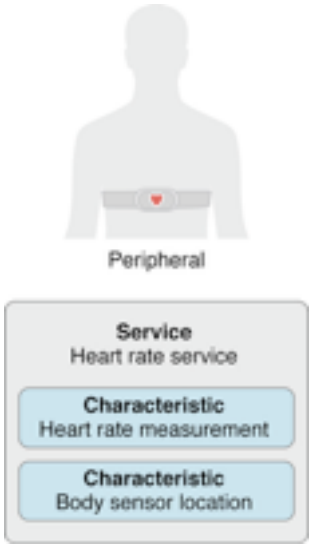
Fonte: Apple (Setembro de 2013)

[3] Pacotes onde cada dispositivo faz anúncio de informações básicas, indicando que está disponível para estabelecer conexões BLE

Os dispositivos periféricos podem possuir vários serviços diferentes, não ficando restritos à enviar somente uma informação. Um termostato, por exemplo, poderia enviar a temperatura atual e a leitura registrada nas últimas horas.

Cada serviço disponibilizado por um periférico pode possuir vários tipos de informações, chamados de *characteristics*:

Figura 5 - Serviços e características de um periférico



Fonte: Apple (Setembro de 2013)

Como os pacotes de *advertising* citados anteriormente são intencionalmente pequenos, com tamanho máximo 38 Bytes (APPLE) para fins de desempenho, eles não necessariamente listam todas as funcionalidades e informações disponíveis por um periférico. Cabe ao central solicitar a lista de *services* e *characteristics* que o periférico possui.

3 TRABALHOS RELACIONADOS

3.1 Bluetooth Low Energy Mesh Networks: A Survey

Segundo Darroudi (2017), existem dois tipos de soluções para a criação de redes mesh sobre bluetooth, uma utilizando técnicas de *flooding*, e outra utilizando técnicas de roteamento.

Flooding é a técnica mais simples de ser implementada: cada nó envia mensagens broadcast para toda a rede. Estes pacotes são reenviados por todos os

nós no caminho, para todos os seus vizinhos. As únicas verificações necessárias neste modelo são:

- TTL: Contador que é decrementado à cada salto do pacote, e faz com que o pacote seja descartado quando o primeiro chegar à zero;

- Identificador de destino: Se o pacote atinge seu destino, o nó será responsável por processar o pacote e não o irá retransmitir;

- Verificação de transmissão: Cada pacote retransmitido tem seu identificador único salvo em cada nó. Desta forma, um nó irá verificar se já enviou determinado pacote, anulando sua retransmissão.

A técnica de roteamento se subdivide em duas versões: roteamento estático e roteamento dinâmico. Na primeira, os nós são organizados em uma estrutura em forma de árvore, onde os nós no nível mais alto da árvore se comportam como mestres dos nós inferiores (escravos); quando um nó deseja alcançar outro integrante da rede, ele consulta o seu nó mestre, que propaga a consulta pela hierarquia da árvore até encontrar o nó raiz. O nó raiz, por sua vez, estende a consulta por seus nós escravos no sentido oposto, até encontrar o destino. O problema desta abordagem é que se um nó ficar indisponível, pode comprometer a comunicação de uma grande parte da rede, pois não há suporte a rotas alternativas.

Na técnica de roteamento dinâmico, se o nó que deseja enviar um pacote não conhece a rota para o destino, ele envia uma mensagem *broadcast* para seus nós vizinhos, que repetem esta operação até que algum nó que conhece o destino indique sua rota. Os pacotes possuem valor TTL[4] definido na origem, bem como tempo máximo até a descoberta de rota.

A Tabela 1, construída durante este estudo (Darroudi, 2017), classifica os diversos tipos de redes mesh estudados e ilustra a falta de um padrão definido quanto ao seu funcionamento:

Tabela 1 - Principais características das soluções para redes mesh

	Proposal Name	**Year**	**Multi-Hop Paradigm**	**Bluetooth Version**	**Type of Channels**	
					Advertising Channels	**Data Channels**
Flooding	N/A	2016	Trickle + gossiping	4.0	Data	-
	BLEmesh	2015	Bounded flooding	4.2	Data	-

[4] Time To Live. Define o número máximo de nós pelo qual um pacotes pode ser propagado

Routing	**Static**	N/A	2014	Tree-based routing	4.0	-	Data
		RT-BLE	2016	Pre-configured	4.1	-	Data
	Dynamic	MHTS	2013	On-demand routing	4.0	Routing	Routing/Data
		BMN	2015	DAG-based routing	4.1	Routing	Data
		N/A	2015	On-demand routing	4.1	-	Routing/Data
		N/A	2015	Named Data Networking	4.1	-	Routing/Data
		ALBER	2016	DAG-based routing	4.1	Routing	Data

Fonte: Darroudi (2017)

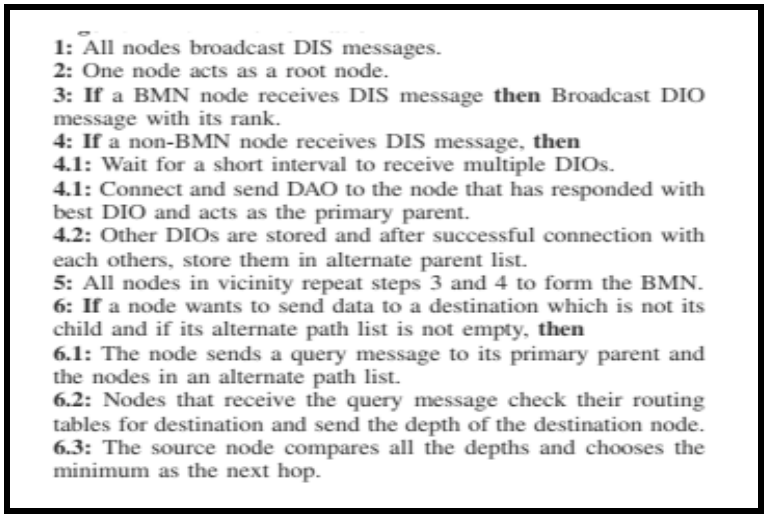
3.2 Uma rede mesh para dispositivos móveis usando Bluetooth LE

Segundo Sirur (2015), na maioria das técnicas de roteamento em redes sem fio, assume-se que os nós são estáticos e não alteram sua posição até o final da conexão, o que não é uma pressuposição realista.

Para contornar este problema, foi proposta uma rede bluetooth mesh para dispositivos não-estáticos, utilizando-se uma técnica de roteamento para a criação da rede, pois segundo os integrantes do estudo esta abordagem gera menos consumo de energia dos dispositivos envolvidos que as técnicas de *flooding*.

O roteamento foi criado utilizando-se uma estrutura mestre/escravo e seu algoritmo básico de formação pode ser visto na Figura 6:

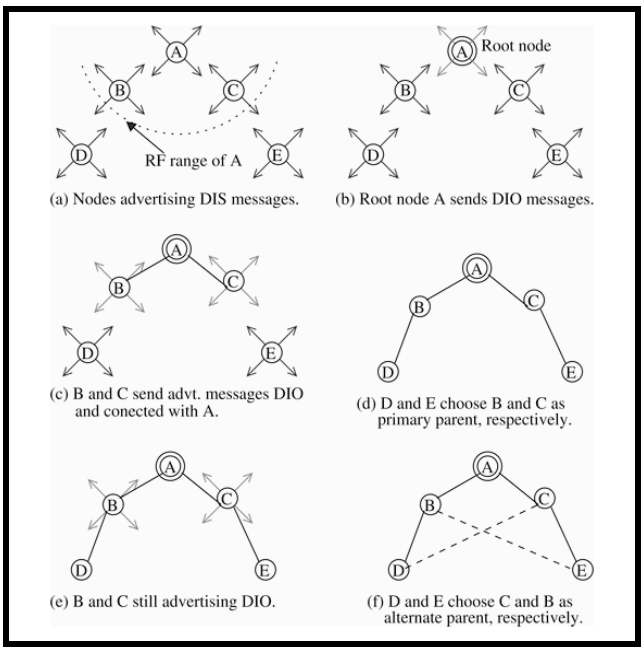
Figura 6 - Algoritmo de formação da BMN


1: All nodes broadcast DIS messages.
2: One node acts as a root node.
3: If a BMN node receives DIS message **then** Broadcast DIO message with its rank.
4: If a non-BMN node receives DIS message, **then**
4.1: Wait for a short interval to receive multiple DIOs.
4.1: Connect and send DAO to the node that has responded with best DIO and acts as the primary parent.
4.2: Other DIOs are stored and after successful connection with each others, store them in alternate parent list.
5: All nodes in vicinity repeat steps 3 and 4 to form the BMN.
6: If a node wants to send data to a destination which is not its child and if its alternate path list is not empty, **then**
6.1: The node sends a query message to its primary parent and the nodes in an alternate path list.
6.2: Nodes that receive the query message check their routing tables for destination and send the depth of the destination node.
6.3: The source node compares all the depths and chooses the minimum as the next hop.

Fonte: Sirur (2015)

A Figura 7 exibe a estratégia de contingente caso algum nó perca a conexão com a rede e seja necessário encontrar uma nova rota. Quando um nó filho é adicionado ao nó raiz, este escolhe dentre os filhos um nó que será seu “sucessor”, caso este venha a sair da rede. Esta escolha é baseada em alguns parâmetros, como peso da sub-árvore, número de nós na sub-árvore, potência residual, entre outros.

Figura 7 – Estratégia de contingente



Fonte: Sirur (2015)

4 METODOLOGIA

4.1 Metodologia da pesquisa

Foi realizada uma pesquisa bibliográfica para elaboração inicial de um protótipo para a prova de conceito, composto por um aplicativo simples de envio de pacotes Bluetooth LE, e outro aplicativo para o recebimento do sinal. Estes dois aplicativos foram desenvolvidos em linguagem Swift com o intuito de elucidar as limitações do padrão BLE na plataforma iOS.

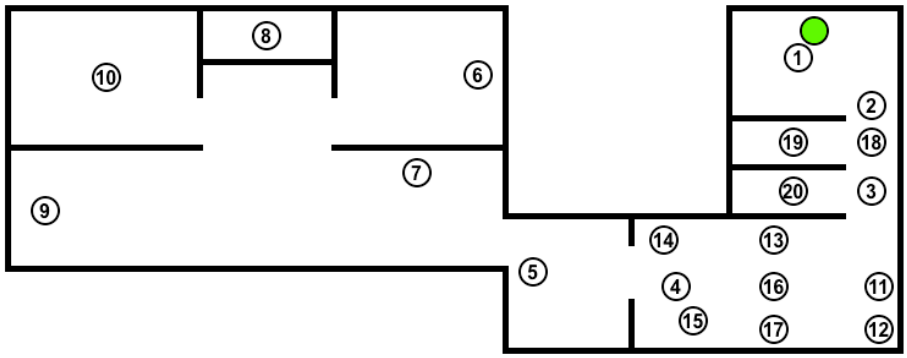
O aplicativo de envio de sinal foi instalado em um aparelho iPhone 6S e o aplicativo de recebimento de sinal foi instalado em um iPad mini 2. Optou-se por fazer o envio de sinal a partir de um iPhone pois este possui chip de rádio de menor

potência que o do iPad, o que forneceria um retrato do pior caso, visto que se deseja conhecer as limitações gerais do sistema.

Como técnica de coleta de dados, foram realizados dois experimentos para conhecer o padrão de atenuação do sinal BLE em função da distância entre dois nós: o primeiro sendo *indoor* e o segundo em campo aberto.

No primeiro experimento foi traçado um mapa de calor do sinal BLE, captado em um local com estrutura de construção mista (paredes de alvenaria, paredes de vidro e portas de madeira).

Figura 8 - Mapa de captura dos valores RSSI



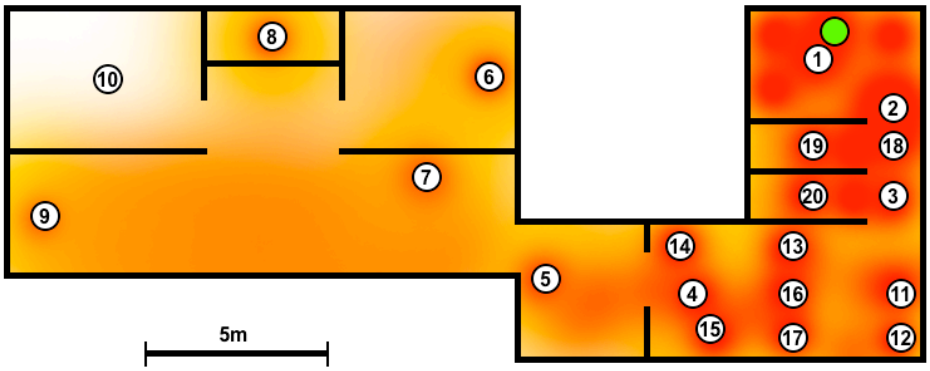
Fonte: Elaborado pelo autor

Tabela 2 - RSSI *indoor*

Posição	RSSI	Posição	RSSI
1	-40	11	-95
2	-69	12	-87
3	-78	13	-96
4	-99	14	-101
5	-94	15	-93
6	-87	16	-96
7	-99	17	-91
8	-103	18	-71
9	-99	19	-78
10	Sem conexão	20	-91

Fonte: Elaborado pelo autor

Figura 9 - Mapa de calor *indoor*

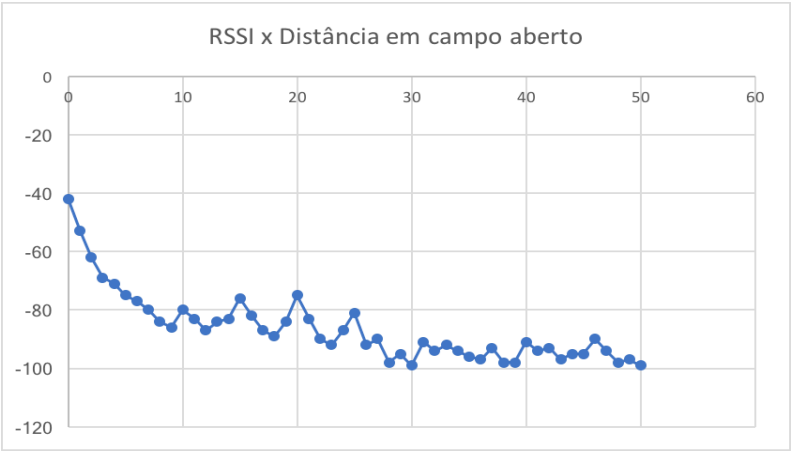


Fonte: Elaborado pelo autor

O dispositivo emissor foi posicionado no ponto verde, como pode ser visto na Figura 9. Os outros pontos indicam os locais de medição do sinal pelo receptor, com seus valores registrados na Tabela 2.

No segundo experimento, foi realizada a leitura do valor RSSI entre o dispositivo emissor e receptor num trecho de 50m em campo aberto, em intervalos de 1m. Foi possível observar, em concordância com Gao (2015), que os valores RSSI lidos se tornavam mais instáveis com o aumento da distância entre os dispositivos, fazendo com que fosse necessário registrar o valor da moda observada – em alguns momentos a variação chegou à mais de 10 decibéis.

Gráfico 1 - RSSI x Distância



Fonte: Elaborado pelo autor

Os valores registrados no segundo experimento podem ser observados em sua totalidade no Apêndice A.

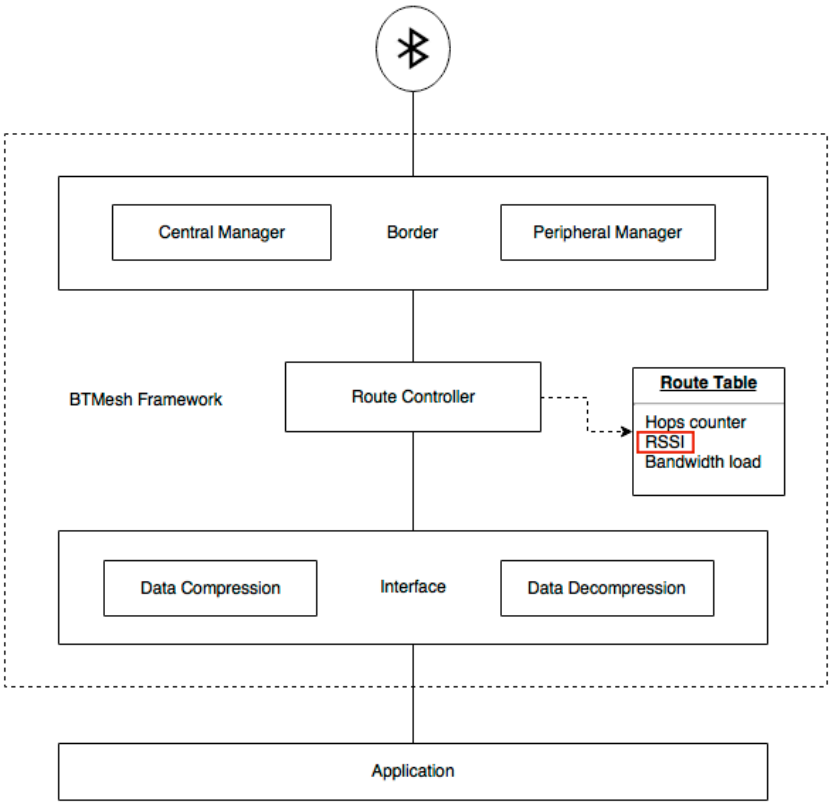
4.2 Metodologia do desenvolvimento

Uma vez de posse de conhecimento básico quanto à influência da distância entre os dispositivos e elementos físicos do ambiente (paredes, árvores, etc) sobre a leitura RSSI, foi possível iniciar o desenvolvimento do framework em si.

4.2.1 Arquitetura

Na figura 10 é possível observar o esquema da arquitetura básica proposta para o framework BTMesh. O coração do framework é o controlador de rotas (*Route Controller*), que irá tomar a decisão de qual rota seguir baseando-se nas métricas salvas em sua tabela de roteamento (*Route Table*).

Figura 10 - Arquitetura básica do framework BTMesh



Fonte: Elaborado pelo autor

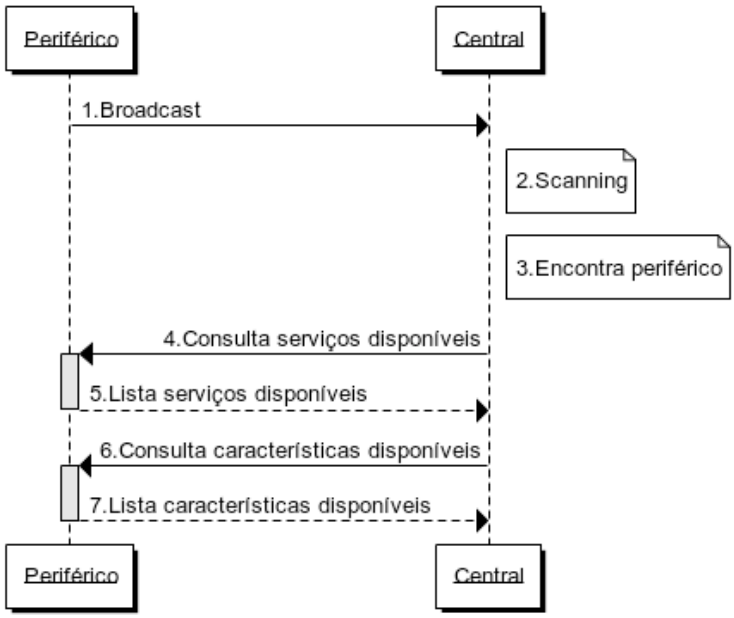
As métricas são compostas pelo número de saltos necessário para se atingir um determinado nó, além do valor RSSI total do trecho – composto pela soma de todos os valores RSSI[5] entre os nós envolvidos – e a taxa de ocupação de banda nos nós envolvidos (*Bandwidth load*).

Existe ainda uma camada de borda (*Border*), responsável por fazer a comunicação com o framework Core Bluetooth, através de suas classes *Central Manager* e *Peripheral Manager*, e uma camada que serve de interface para as aplicações do usuário, onde é realizada a compactação e descompactação dos dados trafegados à fim de otimizar o desempenho do sistema.

4.2.2 Camada de borda

Conforme descrito no item 2.4.4.1, a comunicação BLE se dá, entre dois dispositivos, através dos chamados “Central” e “Peripheral”. Porém, para a transmissão de informações efetiva entre dois aparelhos, ambos devem seguir um determinado fluxo de ações. Esta sucessão de eventos é descrita em detalhes no Quadro 1 e pode ser vista no diagrama de sequência apresentado a seguir:

Figura 11 – Diagrama de sequência BLE



Fonte: Elaborado pelo autor

[5] Atualmente, apenas o valor RSSI está sendo utilizado como métrica, como será visto no decorrer do trabalho.

Quadro 1 - Escrita de valor em característica

Passo	Ação
1	A central recebe o valor da característica desejada via notificação .Inicialmente, quando um periférico deseja ser visto pelo elemento/dispositivo, ele realiza o envio de pacotes “broadcast” para anunciar que possui informações disponíveis. Este pacote possui informações básicas do periférico, como seu identificador único de periférico (UUID – *Universallly Unique Identifier*) e identificador único dos seus serviços disponíveis.
2	Uma central que deseja ler ou escrever informações em periféricos precisa iniciar um ciclo de *scanning*, a fim de detectar periféricos BLE nas proximidades. Esta procura pode ser configurada para acontecer de duas maneiras: na primeira, a busca é “cega”, seu seja, a central vai fazer *scanning* para encontrar qualquer periférico que estiver fazendo broadcast no momento – sendo necessário posteriormente filtrar os resultados para encontrar os serviços desejados –; na segunda, a busca pode ser configurada para encontrar apenas periféricos que disponibilizem algum tipo específico de serviço (que pode ser representado por seu identificador único). Neste segundo formato, o tempo necessário para encontrar dispositivos, bem como o consumo de bateria, são consideravelmente menores.
3 e 4	Uma vez encontrado um ou mais periféricos que atendam às expectativas da central solicitante, esta inicia uma solicitação da lista completa dos serviços disponíveis por este periférico. Os serviços presentes nos pacotes de *advertising* podem não representar a totalidade de serviços disponibilizados, uma vez que tais pacotes possuem tamanho limitado que pode ser reduzido dinamicamente pelo sistema operacional, para garantir que outros aplicativos atuando simultaneamente no mesmo dispositivo também consigam fazer *advertising*
5	Periférico lista todos os seus serviços disponíveis, através de seus identificadores únicos.
6	De posse do serviço desejado, a central inicia uma solicitação pela lista de características presentes em tal serviço. Novamente, a busca pode ser “cega”, listando todas as características disponíveis, ou pode ser específica por um ou mais serviços desejados, o que agiliza o retorno e preserva a bateria do dispositivo.
7	O periférico lista todas as suas características disponíveis, através de seus identificadores únicos.

Fonte: Elaborado pelo autor.

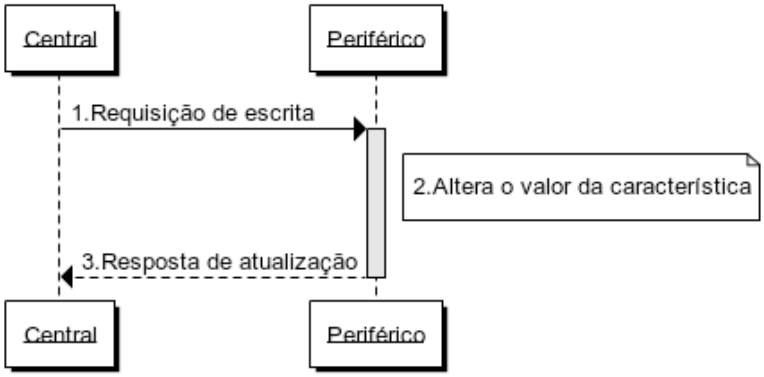
As características podem ser, basicamente, de 3 tipos: leitura, escrita ou notificação (leitura).

No caso de uma característica do tipo escrita, basta que a central solicite a modificação do valor presente em tal característica. Se a solicitação de escrita não tiver sido configurada para receber confirmação, a escrita será feita empregando-se a política de *best effort* (similar ao protocolo UDP, em comparação com TCP)[6]. Se a

[6] Pacotes UDP não retornam, para a origem, confirmação de entrega ao destinatário, ao contrário dos pacotes TCP que possuem tal confirmação.

solicitação de escrita for realizada com expectativa de confirmação, o periférico irá notificar a central de que a escrita foi ou não realizada com sucesso.

Figura 12 – Escrita de valor em característica



Fonte: Elaborado pelo autor

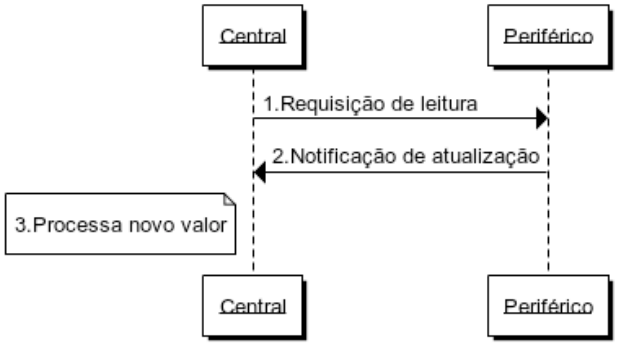
Quadro 2 - Escrita de valor em característica

Passo	Ação
1	A central requisita escrita em uma característica, informando novo valor.
2	O periférico altera o valor da característica.
3	(opcional) O periférico informa a central de que a escrita foi ou não bem-sucedida.

Fonte: Elaborado pelo autor.

As características do tipo notificação são utilizadas para informar à central que fez subscrição de leitura de que seu valor foi alterado. São utilizadas para situações em que tais valores mudam dinamicamente.

Figura 13 – Leitura de característica dinâmica



Fonte: Elaborado pelo autor

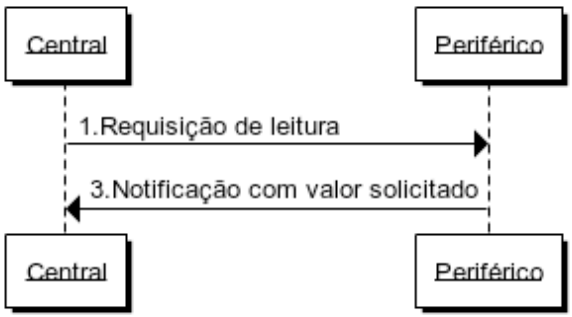
Quadro 3 - Escrita de valor em característica

Passo	Ação
1	A central faz requisição de leitura de uma característica do tipo notificação, que é atualizada dinamicamente.
2	O periférico notifica a central subscrita de que o valor da característica foi alterado.
3	A central recebe o novo valor e pode utilizá-lo.

Fonte: Elaborado pelo autor.

As características do tipo leitura, como o próprio nome sugere, são utilizadas para leitura de valores de um dispositivo. Mas esta leitura não acontece instantaneamente; quando é realizada uma solicitação de leitura, a central que originou tal solicitação é notificada do valor da característica em questão, assim que possível, de forma similar ao que acontece com as características do tipo notificação, mas com a diferença que esta notificação ocorre apenas uma vez.

Figura 14 – Leitura de valor em característica



Fonte: Elaborado pelo autor

Quadro 4 - Escrita de valor em característica

Passo	Ação
1	A central faz requisição de leitura de característica.
2	O periférico envia uma notificação com o valor da característica.
3	A central recebe o valor da característica desejada via notificação[7].

Fonte: Elaborado pelo autor.

[7] Na verdade, a notificação não carrega o valor da modificação de tabela em si, mas sim um indicador da mudança, como será visto no capítulo 6.1

4.2.3 Boas práticas na utilização do Core Bluetooth

Durante o desenvolvimento do BTMesh, alguns detalhes do funcionamento do framework Core Bluetooth foram desvendados e entendidos, sendo de vital importância para o seu bom funcionamento. Alguns exemplos:

Quando, por exemplo, uma central se inscreve para receber notificações de um periférico, a informação da notificação pode ser enviada na própria chamada de notificação. Mas isto só é verdade para pequenas porções de informação (de até 38 Bytes). Se formos trafegar dados maiores, como por exemplo a tabela de roteamento de um dispositivo que alcança um grande número de nós, não é possível realizar isso diretamente no corpo da notificação. Neste caso, podemos enviar apenas uma notificação com algum tipo de identificador e escrever o valor que se deseja transmitir em uma característica do tipo leitura, que pode ser requisitada pela central posteriormente. As características do tipo leitura possuem recurso de envio dos dados em partes, que podem ser remontadas no destino; que é uma abordagem mais indicada para este tipo de situação.

Foi constatado também que algumas vezes os periféricos são momentaneamente desconectados sem razão aparente. Nestes casos, quando a central detecta a desconexão de um periférico, pode-se disparar o *scanning* por um tempo específico, a fim de encontrar novamente o dispositivo perdido. Não é possível simplesmente tentar conectar-se com o periférico perdido, mesmo se for mantida uma referência em memória para o objeto do mesmo, pois este irá mudar seu identificador único na próxima conexão.

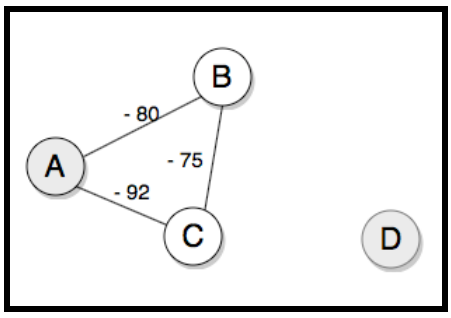
4.3 Roteamento

O roteamento proposto é do tipo dinâmico, ou seja, sempre que há alteração nas conexões entre os nós, estes informam seus nós vizinhos, que informam os demais, sucessivamente em um efeito cascata. Para tal, cada nó, quando estabelece comunicação com outro nó, guarda em sua tabela de roteamento o valor RSSI do nó em questão e a lista de todos os nós vistos por ele, bem como o RSSI total para atingi-los.

Isto é feito através de características notificáveis nos periféricos de cada nó. Quando um nó se conecta com o periférico de outro, a central do primeiro se inscreve para receber notificações do serviço de atualização de rotas.

No exemplo a seguir, Figura 15, o nó “E” ainda não faz parte da rede mesh composta pelos nós “A”, “B” e “C”. A tabela de roteamento do nó “A” pode ser vista na tabela 3.

Figura 15 – Configuração inicial da rede



Fonte: Elaborado pelo autor

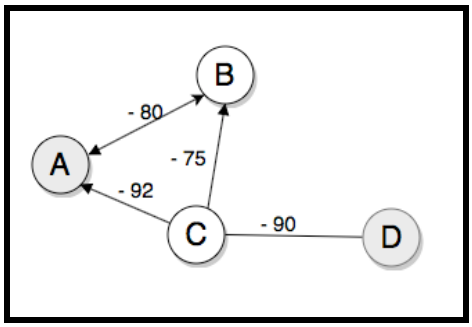
Tabela 3 – Tabela de roteamento do nó “A” antes da chegada do nó “D”

Destino	Nó de saída	**RSSI total**
B	B	-80
B	C	-167
C	C	-92
C	B	-155

Fonte: Elaborado pelo autor

Quando o nó “D” se conecta ao nó “C”, o último notifica seus nós previamente conectados a respeito desta nova conexão. O fluxo desta atualização pode ser visto na figura 16. Após a conexão, o nó “A” pode alcançar “D” tanto por “B” quanto por “C”, apesar dos diferentes valores de RSSI total entre os links.

Figura 16 – Atualização das tabelas de roteamento



Fonte: Elaborado pelo autor

Tabela 4 – Tabela de roteamento do nó “A” depois da chegada do nó “D”

Destino	Nó de saída	**RSSI total**
B	B	-80
B	C	-167
C	C	-92
C	B	-155
D	B	-245
D	C	-182

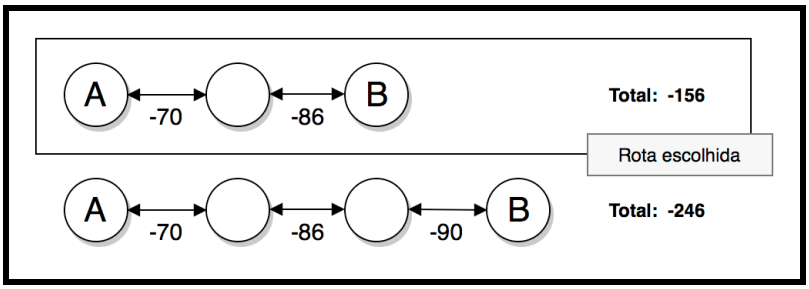
Fonte: Elaborado pelo autor

Quando um nó deseja alcançar outro, consulta em sua tabela de roteamento se o nó desejado está presente. Caso esteja, verifica por quais possíveis caminhos este nó pode ser alcançado e elege a rota que possuir o menor valor RSSI total entre eles.

Inicialmente foi pensado que o número de saltos para atingir um nó seria a métrica principal de avaliação, seguida do valor RSSI caso houvesse mais de uma rota possível com o mesmo número de saltos. Mas esta abordagem poderia favorecer, por exemplo, uma rota de 4 saltos com valor RSSI muito ruim – e com grande chance de desconexão –, em detrimento de uma rota com 5 saltos e valor RSSI maior. Foi observado no estudo prático da seção 4.1 que quando uma conexão fica com valor RSSI abaixo de -100db, a chance de desconexão é muito grande.

Em uma situação normal, a escolha da rota baseando-se no RSSI total do trecho é equivalente a fazer-se esta escolha baseando-se no número de saltos, como pode ser visto na Figura 17:

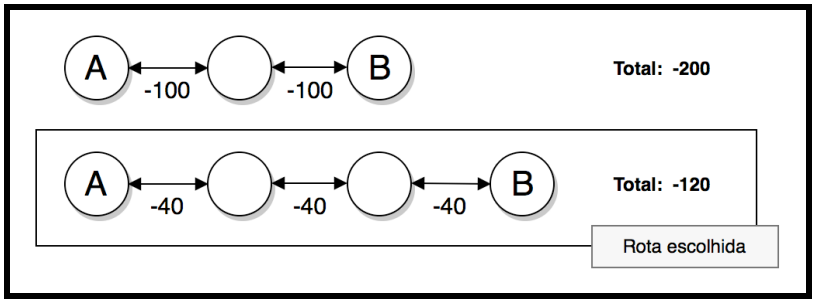
Figura 17 – Situação comum de escolha de rota



Fonte: Elaborado pelo autor

Como o valor RSSI não fica acima de -40db (no caso dos dispositivos estarem lado-a-lado), e dificilmente fica abaixo de -100db (~50m de distância em campo aberto), tendo na maioria dos casos valores por volta de -75db, a decisão pela melhor rota em função do RSSI irá entregar uma melhor solução para os casos extremos, como pode ser visto na Figura 18:

Figura 18 – Situação extrema de escolha de rota



Fonte: Elaborado pelo autor

Esta é uma inferência criada pelo autor, baseada em sua percepção quando à possíveis cenários de uso. Caberia aqui a criação de um experimento futuro para tentar traçar uma relação entre a largura de banda máxima possível de ser trafegada nos nós BLE e variações de sinal RSSI entre os mesmos, à fim de validar esta pré-suposição.

5. ANÁLISE DOS DADOS

A comunicação e o roteamento dos dados entre os nós foi realizada e testada entre três dispositivos disponíveis: um iPad e dois iPhones. Para validar o

funcionamento do roteamento em uma maior malha de nós, foram criados testes simulados com o intuito de validar seu comportamento.

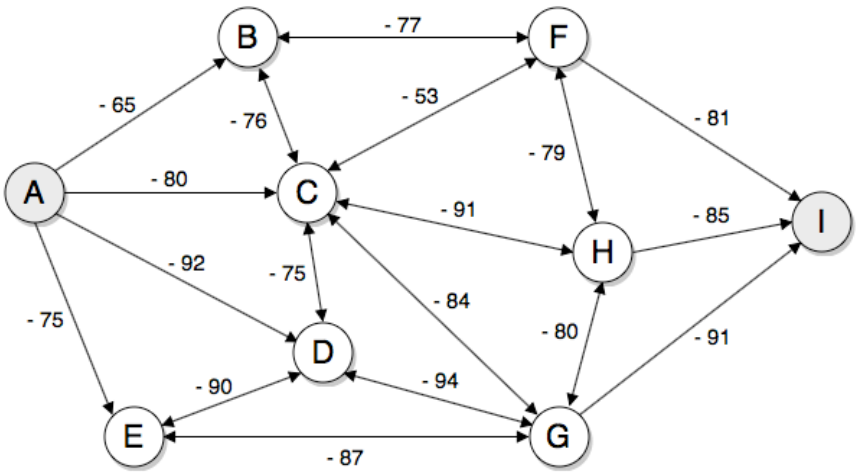
No momento, não está sendo avaliada a performance do roteamento quanto à taxa de transmissão de dados ou variações no consumo de bateria. Estes são itens que certamente serão tratados no desenvolvimento futuro do projeto. Neste momento, o foco é a capacidade de envio das informações dentro da rede, seguindo o melhor caminho.

5.1 Teste unitário para capacidade de decisão pela melhor rota

A fim de testar a capacidade de decisão da camada de roteamento em relação a que caminho seguir entre os nós disponíveis, mesmo sem dispor de um número razoável de dispositivos, foi desenvolvido um teste unitário para avaliar sua assertividade.

Inicialmente foi definida uma situação hipotética na qual nove nós estão interconectados na rede mesh, com o valor RSSI entre eles definido de forma aleatória, conforme a Figura 19:

Figura 19 – Disposição dos nós da simulação



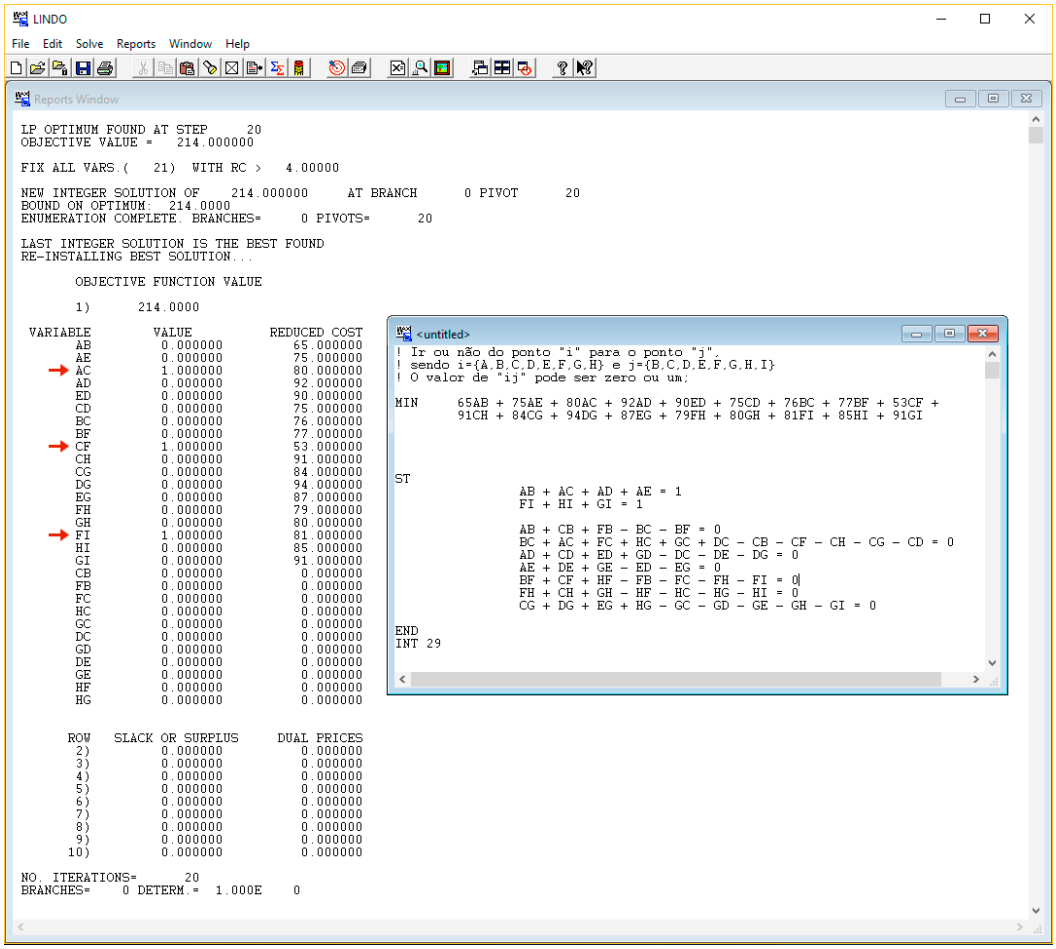
Fonte: Elaborado pelo autor

Note-se que as conexões do ponto “A” são apenas de saída e que as do ponto “I” são apenas de entrada. Todas as outras são bidirecionais, podem ocorrer em qualquer sentido.

A seguir, foi criado um modelo de programação linear que representasse o objetivo desejado, ou seja, descobrir a melhor rota entre o ponto “A” e o ponto “I”. Este modelo foi introduzido no software de programação linear LINDO[8]. Os valores RSSI tiveram seu sinal removido na simulação, pois o tipo de função objetivo era do tipo mínimo, com intenção de encontrar a menor rota, portanto favorecendo valores menores de RSSI entre cada trecho.

O resultado encontrado foi a rota A -> C -> F -> I, como pode ser visto na figura a seguir:

Figura 20 – Resolução do software LINDO



Fonte: Elaborado pelo autor

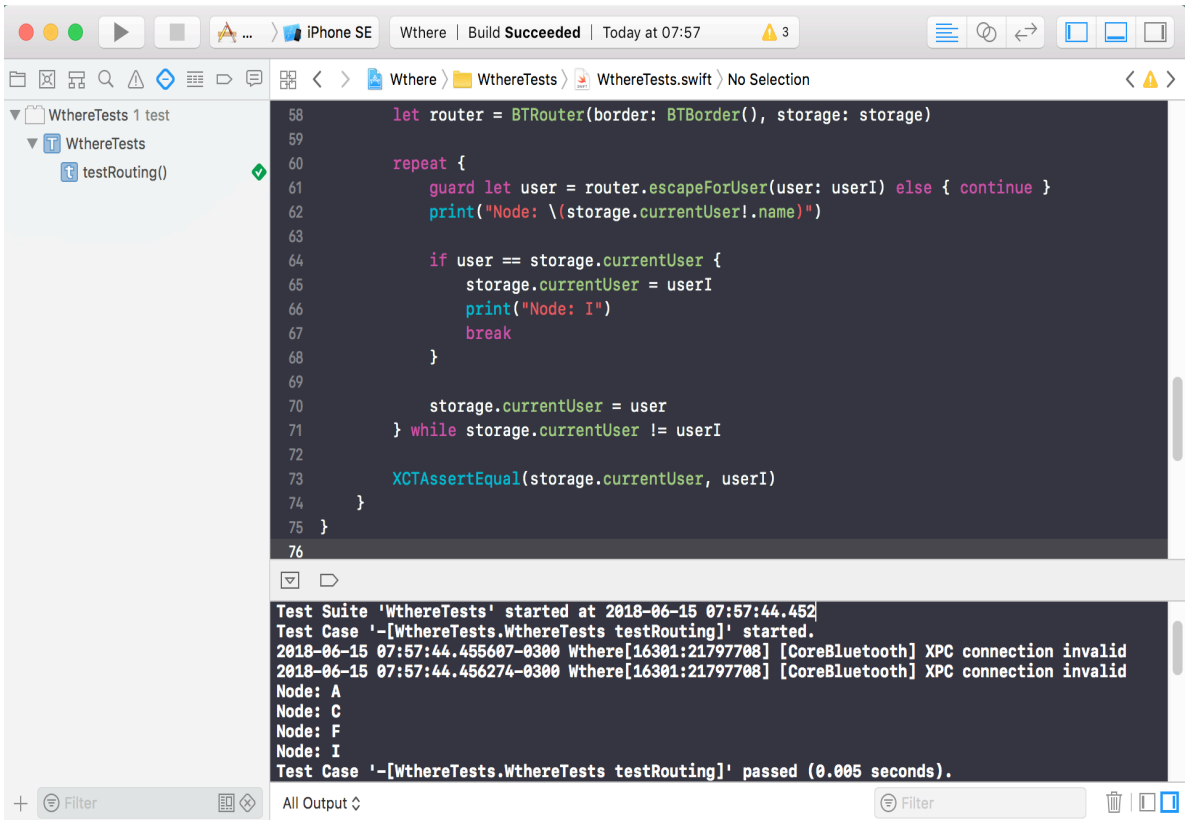
A modelagem e o resultado completo obtido podem ser vistos nos apêndices B e C, respectivamente.

Em seguida, foi desenvolvido o teste unitário, que pode ser visto na íntegra no apêndice D.

[8] LINDO Systems Inc. Disponível: <https://www.lindo.com>. Acesso em: 8 de maio de 2018.

Este teste simulou a mesma organização de nós introduzida na simulação anterior, encontrando a mesma rota sugerida: A -> C -> F -> I. O resultado do teste pode ser visto na Figura 21. O código completo deste teste unitário está exposto no apêndice D.

Figura 21 – Resultado do teste unitário



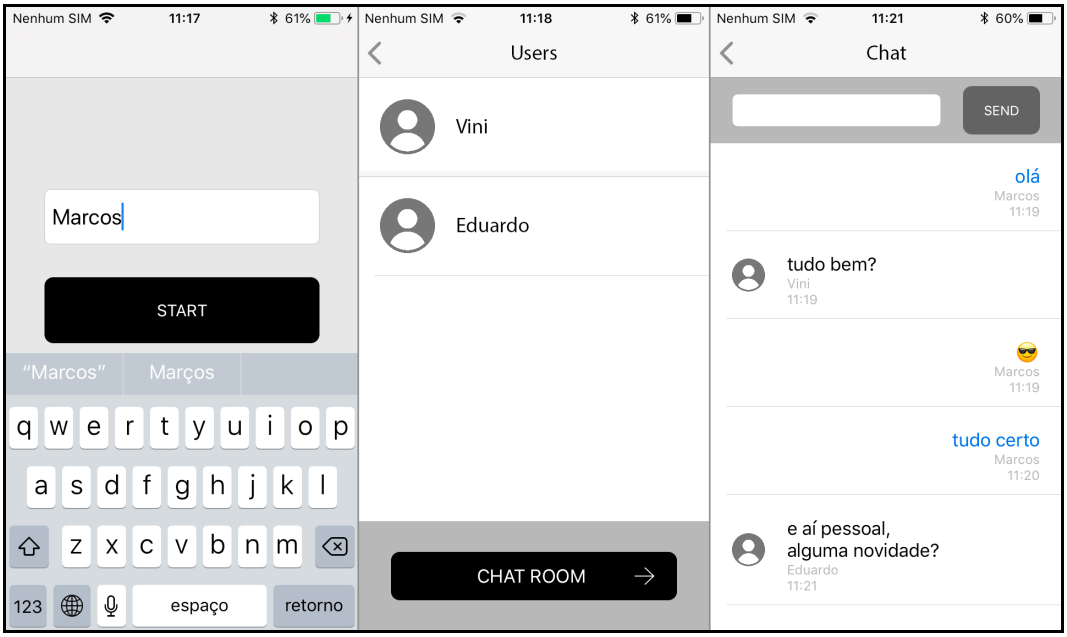
Fonte: Elaborado pelo autor

5.2 Aplicativo de demonstração

Para demonstrar a funcionalidade do framework foi desenvolvido e disponibilizado, juntamente com BTMesh, um aplicativo de demonstração que ensina como utilizá-lo. Este aplicativo chama-se BTChat e possibilita que várias pessoas nas proximidades possam conversar em uma sala virtual de bate-papo através de conexão Bluetooth, sem a necessidade do uso da Internet.

Naturalmente, o aplicativo utiliza-se das vantagens de uma rede mesh, ou seja, usuários que não estejam dentro do alcance de uma conexão Bluetooth tradicional poderão ser alcançados caso haja outros usuários no caminho, por onde a conexão possa ser roteada.

Figura 22 – Aplicativo BTChat Demo



Fonte: Elaborado pelo autor

6. CONCLUSÃO

O desenvolvimento de aplicações em BLE para plataformas móveis se mostrou mais desafiador que o inicialmente previsto. Mesmo com o auxílio de um framework de alto nível como o Core Bluetooth, existe a necessidade de uma grande orquestração de chamadas de serviço apenas para se estabelecer comunicação entre dois nós da forma mais básica possível. A própria natureza móvel do sistema operacional, com foco na economia de energia e compartilhamento de recursos, cria limitações quanto ao que pode ser ou não realizado.

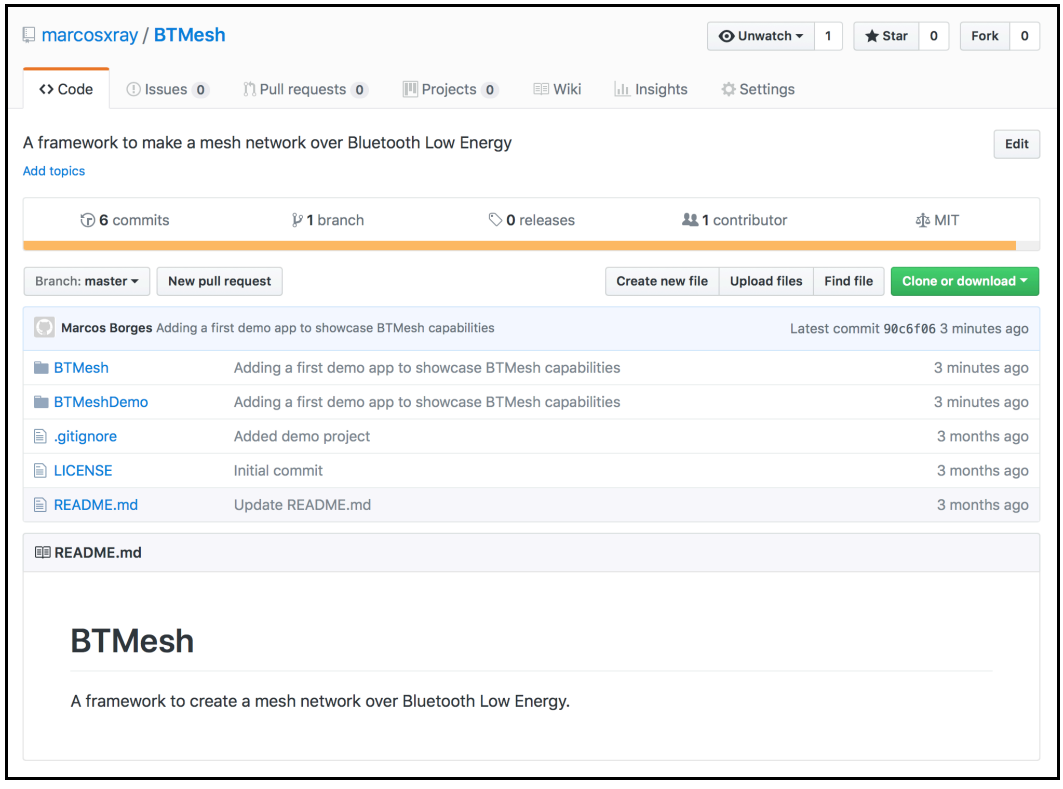
Adicionar a isto o fato de ser necessário desenvolver códigos capazes de contornar e resolver possíveis situações problemáticas e de instabilidade, agregado à complexidade exigida pelo roteamento necessário para a criação da rede mesh, faz com que o projeto siga seu desenvolvimento mesmo após a conclusão deste artigo. Como o BTMesh será disponibilizado como código livre, espera-se que a comunidade de desenvolvimento de software auxilie na sua continuidade.

O presente trabalho contemplou apenas o desenvolvimento para a plataforma iOS, mas como o padrão Bluetooth LE possui implementação em outras plataformas, como por exemplo Android e Windows Phone, seria possível transportá-lo para estas e fazer use de redes mesh entre os mais diversos dispositivos.

Segundo KOLDERUP (2017), a especificação Bluetooth recentemente adicionou suporte a redes mesh. Mas isto de forma alguma reduz a significância do presente trabalho, visto que os novos recursos podem demorar, ou talvez até mesmo não chegar, a serem disponibilizados para os desenvolvedores. Como o foco deste trabalho é em desenvolvimento para plataformas móveis, mais especificamente iOS, vale lembrar que os fabricantes de tais plataformas possuem o controle do que pode ou não ser desenvolvido para elas, através dos frameworks nativos disponibilizados e de regras de publicação e distribuição de aplicativos em suas lojas eletrônicas. Atualmente não é possível acessar os recursos BLE no sistema iOS sem fazer uso do framework Core Bluetooth, e tal framework ainda não suporta redes mesh nativamente. Não se sabe “se” e “quando” isto irá ocorrer, deixando espaço para pesquisa e desenvolvimento independentes, como no caso do BTMesh.

O framework BTMesh está disponível como código-livre no seguinte endereço: <https://github.com/marcosxray/BTMesh>.

Figura 23 – Repositório do projeto BTMesh



Fonte: Elaborado pelo autor

6.1 Desenvolvimentos futuros

Atualmente, quando ocorre alguma modificação na tabela de roteamento de um dos dispositivos envolvidos na rede mesh, este propaga toda a sua tabela para os demais nós. Como visto anteriormente, uma notificação é enviada para indicar aos outros nós que disparem uma requisição de leitura do novo valor. Mas esta notificação em si pode conter alguma informação, mesmo que pequena, como um *hash*.

Neste caso, seria possível criar um histórico dos estados da tabela de roteamento, utilizando um *hash* como índice das modificações (similar ao funcionamento de um repositório do tipo GIT). Ao notificar os outros nós quanto à uma mudança, seria enviado na notificação o *hash* da mesma. Quando um outro nó então solicitasse a leitura do novo estado, passaria em primeiro lugar o *hash* recebido e seria verificado no histórico quais modificações ocorreram após aquele *hash* e então enviadas somente as últimas mudanças. Desta forma seria possível reduzir sensivelmente a quantidade de informação de roteamento trocada entre os nós, liberando mais espaço para a transmissão de dados.

Outro ponto importante de desenvolvimento futuro é a questão de como a comunicação irá se comportar nos diferentes estados da aplicação (primeiro plano, segundo plano, aplicativo fechado, etc.).

No ponto atual de desenvolvimento, apenas está sendo tratado o estado da aplicação ativa em primeiro plano. O comportamento do framework Core Bluetooth bem como do sistema operacional muda significativamente quando um aplicativo se encontra em segundo plano, ou *background*.

Quando o aplicativo se encontra em segundo plano, tanto o central quanto o periférico do dispositivo têm seu comportamento modificado pelo sistema operacional. No caso do central, o iOS mantém ativos os callbacks BLE, informando à aplicação sempre que um periférico é encontrado ou conectado. No entanto, quanto ao *scanning* de outros dispositivos, como podem haver outros aplicativos rodando ao mesmo tempo e utilizando recursos BLE, o tempo de *scanning* vai ser compartilhado (chaveado) entre todos que utilizem tal recurso, resultando em maior tempo de descoberta. Para o *peripheral* a situação não é muito diferente, tendo seu tempo de *advertising* compartilhado com demais aplicações que utilizem tal recurso. Outro detalhe é que os identificadores dos serviços presentes no *advertising* ficam

em uma área de overflow, onde só podem ser vistos por aplicações que estejam fazendo buscas específicas por seus valores. Uma vez que uma parte do aplicativo seja invocada por um callback BLE, quando em *background*, esta tem 10 segundos para realizar operações, antes que o sistema a coloque novamente em repouso.

Diferentemente de quando está em repouso, quando o aplicativo é fechado pró-ativamente pelo usuário, ele passa a não mais receber callbacks BLE. Mas para estes casos, o framework Core Bluetooth possui recurso de preservação de estado do stack BLE, que pode ser utilizado para reaver o ponto de onde a aplicação estava quando foi encerrada.

Outro ponto importante para ser endereçado futuramente é com relação à encriptação dos dados trafegados. Caso o roteamento proposto utilizasse o canal de *advertising* para identificar os nós dentro da rede mesh, como em alguns exemplos vistos na Tabela 1, o roteamento seria exposto, mesmo que os dados fossem encriptados. Neste caso um atacante poderia comprometer o funcionamento da rede. No BTMesh, uma vez que toda a informação de roteamento será trafegada na camada de dados, bastará encriptá-la para deixar o restante da rede invisível para o mesmo.

Por fim, um fator importante para o desenvolvimento futuro é a inclusão da ocupação de banda de cada nó nas métricas de roteamento, a fim de desviar o tráfego de nós com pouca capacidade de atendimento, aumentando a performance do sistema como um todo.

BTMesh: Framework para a criação de redes mesh sobre Bluetooth LE

Abstract: Nowadays the communication between mobile devices is done mostly over the Internet, which is dependent on carriers' infrastructure. If these systems go down, by electronic fail or overflow, users would be unable to share information. As a solution to this problem, this work has the objective of presenting the research and development of a mobile framework to build mesh networks over Bluetooth Low Energy (BLE). It is being proposed the creation of a dynamic communication net between devices, even if they are not connected to the Internet. To do so has been conducted a research to understand BLE's behavior and limitations, network architectures, and reactive programming, to be able to design a protocol with good performance regarding bandwidth and power consumption. The results that could be seen in the experiments show the proposed frameworks' viability as a communication agent of mobile devices using BLE technology, even being in an early development stage.

Keywords: Bluetooth Low Energy. BLE. Reactive Programming. Mesh Network.

REFERÊNCIAS

APPLE, Dezembro de 2015. Disponível em: <https://developer.apple.com/swift/blog/?id=34>. Acesso em: 8 de outubro de 2017.

APPLE, Outubro de 2015. Disponível em: <https://developer.apple.com/library/content/documentation/General/Conceptual/DevPedia-CocoaCore/Cocoa.html>. Acesso em: 17 de novembro de 2017.

APPLE, Setembro de 2013. Disponível em: <https://developer.apple.com/library/content/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html>. Acesso em: 9 de outubro de 2017.

BANDEIRA, SILVIO; FERNANDES, DAILSON. **Redes de computadores**, Secretaria da Educação de Pernambuco, 2013. Disponível em: <https://sisacad.educacao.pe.gov.br/bibliotecavirtual/bibliotecavirtual/texto/CadernodeINFORedesdeComputadoresRDDI.pdf>. Acesso em: 23 de outubro de 2017.

BLUETOOTH Low Energy (BLE)**.** Cypress Semiconductor Corporation, 2015. Disponível em: <http://www.cypress.com/file/220246/download>. Acesso em: 19 de março de 2018.

BLUETOOTH SIG. Disponível em: < https://www.bluetooth.com>. Acesso em 18 de março de 2018.

CERTIOLOGY. **Network Topology**. Disponível em: <http://www.certiology.com/computing/computer-networking/network-topology.html>. Acesso em 22 de novembro de 2017.

DARROUDI, SEYED MAHDI; GOMEZ, CARLES. **Bluetooth Low Energy Mesh Networks: A Survey**. US National Library of Medicine, National Institute of Health, 2017. Disponível em: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5539726/>. Acesso em: 20 de novembro de 2017.

GAO, VINCENT. **Proximity And RSSI**, 2015. Disponível em: <https://blog.bluetooth.com/proximity-and-rssi>. Acesso em: 19 de novembro de 2017.

KOLDERUP, KEN. **Introducing Bluetooth Mesh Networking**. Bluetoothn SIG, 2017. Disponível em: < https://blog.bluetooth.com/introducing-bluetooth-mesh-networking>. Acesso em: 20 de maio de 2018.

ORLOV, BOHDAN. iOS Architecture Patterns. **Demystifying MVC, MVP, MVVM and VIPER**, 2015. Disponível em: <https://medium.com/ios-os-x-development/ios-architecture-patterns-ecba4c38de52>. Acesso em: 14 de outubro de 2017.

OSHEROVE, ROY. **The Art of unit Testing**. Manning Publications, 2014. 266p

MESHKIT SDK. Disponível em: <https://www.opengarden.com/meshkit.html>. Acesso em: 15 novembro 2017.

PILLET, FLORENT. **RxSwift: Reactive Programming with Swift** - Razerware LLC, 2017. 440p

RAY, BRIAN. **Bluetooth Vs. Bluetooth Low Energy: What's The Difference?** LinkLabs, 2015. Disponível em: <https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy>. Acesso em: 24 de novembro de 2017.

SILVA, PATRÍCIA FREITAS; PENHA, JOSÉ AFONSO MORAES; ALVES, GABRIEL MARCELINO. **Estudo do padrão de projeto Observer no desenvolvimento de softwares utilizando a arquitetura MVC**, Universidade Federal de Santa Catarina, 2009. Disponível em: <http://cpu90.ifc-camboriu.edu.br/criacac/tiki-download_file.php?fileId=123>. Acesso em: 28 de outubro de 2017.

SIRUR, SHRUTHI; JUTURU, PRANEETH; GUPTA, HARI PRABHAT; SERIKAR, PRAMOD REDDY, REDDY, YASWANTH KUMAR, BARAK, SULEKHA, KIM, BONGGON. **A Mesh Network for Mobile Devices using Bluetooth Low Energy** - IEEE, 2015. Disponível em:

<http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=7370451>. Acesso em: 10 de outubro de 2017.

APPLE. StartAdvertising. Disponível em: <https://developer.apple.com/documentation/corebluetooth/cbperipheralmanager/1393252-startadvertising>. Acesso em: 10 de abril de 2018.

WOOLLEY, MARTIN. **Bluetooth 5. Go Faster. Go Further.** Bluetooth SIG. Disponível em: <https://www.bluetooth.com/bluetooth-technology/bluetooth5/bluetooth5-paper>. Acesso em: 20 de maio de 2018.

APÊNDICE A – RSSI X DISTÂNCIA EM CAMPO ABERTO

Distância (m)	RSSI	Distância (m)	RSSI
0	-42	26	-92
1	-53	27	-90
2	-62	28	-98
3	-69	29	-95
4	-71	30	-99
5	-75	31	-91
6	-77	32	-94
7	-80	33	-92
8	-84	34	-94
9	-86	35	-96
10	-80	36	-97
11	-83	37	-93
12	-87	38	-98
13	-84	39	-98
14	-83	40	-91
15	-76	41	-94
16	-82	42	-93
17	-87	43	-97
18	-89	44	-95
19	-84	45	-95
20	-75	46	-90
21	-83	47	-94
22	-90	48	-98
23	-92	49	-97
24	-87	50	-99
25	-81		

Fonte: Elaborado pelo autor

APÊNDICE B – MODELAGEM DO TESTE UNITÁRIO

```
! Ir ou não do ponto "i" para o ponto "j",
! sendo i={A,B,C,D,E,F,G,H} e j={B,C,D,E,F,G,H,I}
! O valor de "ij" pode ser zero ou um;

MIN     65AB + 75AE + 80AC + 92AD + 90ED + 75CD +
        76BC + 77BF + 53CF + 91CH + 84CG + 94DG +
        87EG + 79FH + 80GH + 81FI + 85HI + 91GI
ST
        AB + AC + AD + AE = 1
        FI + HI + GI = 1
        AB + CB + FB - BC - BF = 0
        BC + AC + FC + HC + GC + DC - CB - CF - CH - CG - CD = 0
        AD + CD + ED + GD - DC - DE - DG = 0
        AE + DE + GE - ED - EG = 0
        BF + CF + HF - FB - FC - FH - FI = 0
        FH + CH + GH - HF - HC - HG - HI = 0
        CG + DG + EG + HG - GC - GD - GE - GH - GI = 0
END
INT 29
```

APÊNDICE C – SOLUÇÃO DO SOFTWARE LINDO

LP OPTIMUM FOUND AT STEP 20
OBJECTIVE VALUE = 214.000000

FIX ALL VARS.(21) WITH RC > 4.00000

NEW INTEGER SOLUTION OF 214.000000 AT BRANCH 0 PIVOT 20
BOUND ON OPTIMUM: 214.0000
ENUMERATION COMPLETE. BRANCHES= 0 PIVOTS= 20

LAST INTEGER SOLUTION IS THE BEST FOUND
RE-INSTALLING BEST SOLUTION...

OBJECTIVE FUNCTION VALUE

1) 214.0000

VARIABLE	VALUE	REDUCED COST
AB	0.000000	65.000000
AE	0.000000	75.000000
AC	1.000000	80.000000
AD	0.000000	92.000000
ED	0.000000	90.000000
CD	0.000000	75.000000
BC	0.000000	76.000000
BF	0.000000	77.000000
CF	1.000000	53.000000
CH	0.000000	91.000000
CG	0.000000	84.000000
DG	0.000000	94.000000
EG	0.000000	87.000000
FH	0.000000	79.000000
GH	0.000000	80.000000
FI	1.000000	81.000000
HI	0.000000	85.000000
GI	0.000000	91.000000
CB	0.000000	0.000000
FB	0.000000	0.000000
FC	0.000000	0.000000
HC	0.000000	0.000000

GC	0.000000	0.000000
DC	0.000000	0.000000
GD	0.000000	0.000000
DE	0.000000	0.000000
GE	0.000000	0.000000
HF	0.000000	0.000000
HG	0.000000	0.000000

ROW	SLACK OR SURPLUS	DUAL PRICES
2)	0.000000	0.000000
3)	0.000000	0.000000
4)	0.000000	0.000000
5)	0.000000	0.000000
6)	0.000000	0.000000
7)	0.000000	0.000000
8)	0.000000	0.000000
9)	0.000000	0.000000
10)	0.000000	0.000000

NO. ITERATIONS= 20

BRANCHES= 0 DETERM.= 1.000E 0

APÊNDICE D – CÓDIGO DO TESTE UNITÁRIO

```swift
//  WthereTests.swift

import XCTest
import RxSwift
@testable import Wthere

class WthereTests: XCTestCase {

    override func setUp() {
        super.setUp()
    }

    override func tearDown() {
        super.tearDown()
    }

    func testRouting() {
        let nodeA = NodeFactory.generateNode_A()
        let nodeB = NodeFactory.generateNode_B()
        let nodeC = NodeFactory.generateNode_C()
        let nodeD = NodeFactory.generateNode_D()
        let nodeE = NodeFactory.generateNode_E()
        let nodeF = NodeFactory.generateNode_F()
        let nodeG = NodeFactory.generateNode_G()
        let nodeH = NodeFactory.generateNode_H()
        let nodeI = NodeFactory.generateNode_I()

        let userA = User(node: nodeA)
        let userB = User(node: nodeB)
        let userC = User(node: nodeC)
        let userD = User(node: nodeD)
        let userE = User(node: nodeE)
        let userF = User(node: nodeF)
        let userG = User(node: nodeG)
        let userH = User(node: nodeH)
        let userI = User(node: nodeI)

        let storage = StorageMock()
        storage.addUser(user: userA)
        storage.addUser(user: userB)
        storage.addUser(user: userC)
        storage.addUser(user: userD)
        storage.addUser(user: userE)
        storage.addUser(user: userF)
        storage.addUser(user: userG)
        storage.addUser(user: userH)
        storage.addUser(user: userI)

        storage.currentUser = userA
        let router = BTRouter(border: BTBorder(), storage: storage)

        repeat {
            guard let user = router.escapeForUser(user: userI) else { continue }
            print("Node: \(storage.currentUser!.name)")

            if user == storage.currentUser {
                storage.currentUser = userI
                print("Node: I")
                break
            }

            storage.currentUser = user
        } while storage.currentUser != userI

        XCTAssertEqual(storage.currentUser, userI)
```

```
    }
}

// MARK: - Mocks

class StorageMock: StorageProtocol {
    var currentUser: User?
    var users: BehaviorSubject<Set<User>> = BehaviorSubject<Set<User>>(value: [])

    func addUser(user: User) {
        var users = (try? self.users.value()) ?? []
        users.insert(user)
        self.users.onNext(users)
    }

    func removeUser(user: User) {
        var users = (try? self.users.value()) ?? []
        guard let index = users.index(of: user) else { return }
        users.remove(at: index)
        self.users.onNext(users)
    }
}


//  NodeParameters.swift

let A_IDENTIFIER = "67B49107-E7DC-45E4-B923-1A73936469AB"
let B_IDENTIFIER = "6A2A2765-A09E-41A4-B7FE-177937CA0E7A"
let C_IDENTIFIER = "92F4EAC5-CD9C-45A9-A031-29E2B8FCC29E"
let D_IDENTIFIER = "3B2AF260-E3CA-4DD6-A779-4189AA18726A"
let E_IDENTIFIER = "89988C47-135F-40F7-AEE5-5F2C86138110"
let F_IDENTIFIER = "3B6BC93A-0B9C-4753-B676-E6059CE472F4"
let G_IDENTIFIER = "30355EC6-8798-4CE3-B3B7-E7E27C303A24"
let H_IDENTIFIER = "67815B0E-BB60-4129-A1AC-7606A83FE3AF"
let I_IDENTIFIER = "02EF6906-DCD4-4328-8890-E3635C5C1972"

let AB_RSSI = -65
let AC_RSSI = -80
let AD_RSSI = -92
let AE_RSSI = -75

let BC_RSSI = -76
let BF_RSSI = -77
let BA_RSSI = -65

let CA_RSSI = -80
let CB_RSSI = -76
let CD_RSSI = -75
let CF_RSSI = -53
let CH_RSSI = -91
let CG_RSSI = -84

let DA_RSSI = -92
let DC_RSSI = -75
let DG_RSSI = -94
let DE_RSSI = -90

let EA_RSSI = -75
let ED_RSSI = -90
let EG_RSSI = -87

let FB_RSSI = -77
let FC_RSSI = -53
let FH_RSSI = -79
let FI_RSSI = -81

let HC_RSSI = -91
let HF_RSSI = -79
let HG_RSSI = -80
```

```
let HI_RSSI = -85

let GC_RSSI = -84
let GD_RSSI = -94
let GE_RSSI = -87
let GH_RSSI = -80
let GI_RSSI = -91

let IF_RSSI = -81
let IG_RSSI = -91
let IH_RSSI = -85


//  NodeFactory.swift

import Foundation
import RxSwift
@testable import Wthere

class NodeFactory {

    static func generateNode_A() -> BTNode {
        let node = BTNode(name: "A", identifier: A_IDENTIFIER)

        let item_AB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                                  escapeNodeIdentifier: A_IDENTIFIER,
                                  targetRssi: AB_RSSI,
                                  escapeRssi: 0,
                                  targetName: "B")

        let item_AC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                                  escapeNodeIdentifier: A_IDENTIFIER,
                                  targetRssi: AC_RSSI,
                                  escapeRssi: 0,
                                  targetName: "C")

        let item_AD = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                  escapeNodeIdentifier: A_IDENTIFIER,
                                  targetRssi: AD_RSSI,
                                  escapeRssi: 0,
                                  targetName: "D")

        let item_AE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                                  escapeNodeIdentifier: A_IDENTIFIER,
                                  targetRssi: AE_RSSI,
                                  escapeRssi: 0,
                                  targetName: "E")

        let item_AF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: AC_RSSI + CF_RSSI,
                                  escapeRssi: AC_RSSI,
                                  targetName: "F")

        let item_AG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                                  escapeNodeIdentifier: E_IDENTIFIER,
                                  targetRssi: AE_RSSI + EG_RSSI,
                                  escapeRssi: AE_RSSI,
                                  targetName: "G")

        let item_AH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: AC_RSSI + CH_RSSI,
                                  escapeRssi: AC_RSSI,
                                  targetName: "H")

        let item_AI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: AC_RSSI + CF_RSSI + FI_RSSI,
                                  escapeRssi: AC_RSSI,
```

```
                                targetName: "I")

        node.visibleNodeItems.onNext([item_AB,
                                      item_AC,
                                      item_AD,
                                      item_AE,
                                      item_AF,
                                      item_AG,
                                      item_AH,
                                      item_AI])
        return node
    }

    static func generateNode_B() -> BTNode {
        let node = BTNode(name: "B", identifier: B_IDENTIFIER)

        let item_BA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                                  escapeNodeIdentifier: B_IDENTIFIER,
                                  targetRssi: AB_RSSI,
                                  escapeRssi: 0,
                                  targetName: "A")

        let item_BC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                                  escapeNodeIdentifier: B_IDENTIFIER,
                                  targetRssi: BC_RSSI,
                                  escapeRssi: 0,
                                  targetName: "C")

        let item_BF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                  escapeNodeIdentifier: B_IDENTIFIER,
                                  targetRssi: BF_RSSI,
                                  escapeRssi: 0,
                                  targetName: "F")

        let item_BD = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: BC_RSSI + CD_RSSI,
                                  escapeRssi: BC_RSSI,
                                  targetName: "D")

        let item_BE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                                  escapeNodeIdentifier: A_IDENTIFIER,
                                  targetRssi: BA_RSSI + AE_RSSI,
                                  escapeRssi: BA_RSSI,
                                  targetName: "E")

        let item_BG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: BC_RSSI + CG_RSSI,
                                  escapeRssi: BC_RSSI,
                                  targetName: "G")

        let item_BH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                                  escapeNodeIdentifier: F_IDENTIFIER,
                                  targetRssi: BF_RSSI + FH_RSSI,
                                  escapeRssi: BF_RSSI,
                                  targetName: "H")

        let item_BI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                                  escapeNodeIdentifier: F_IDENTIFIER,
                                  targetRssi: BF_RSSI + FI_RSSI,
                                  escapeRssi: BF_RSSI,
                                  targetName: "I")

        node.visibleNodeItems.onNext([item_BA,
                                      item_BC,
                                      item_BD,
                                      item_BE,
                                      item_BF,
                                      item_BG,
```

```
                                    item_BH,
                                    item_BI])

        return node
    }

    static func generateNode_C() -> BTNode {
        let node = BTNode(name: "C", identifier: C_IDENTIFIER)

        let item_CA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: CA_RSSI,
                                  escapeRssi: 0,
                                  targetName: "A")

        let item_CB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: CB_RSSI,
                                  escapeRssi: 0,
                                  targetName: "B")

        let item_CD = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: CD_RSSI,
                                  escapeRssi: 0,
                                  targetName: "D")

        let item_CF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: CF_RSSI,
                                  escapeRssi: 0,
                                  targetName: "F")

        let item_CG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: CG_RSSI,
                                  escapeRssi: 0,
                                  targetName: "G")

        let item_CH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: CH_RSSI,
                                  escapeRssi: 0,
                                  targetName: "H")

        let item_CE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                                  escapeNodeIdentifier: A_IDENTIFIER,
                                  targetRssi: CA_RSSI + AE_RSSI,
                                  escapeRssi: CA_RSSI,
                                  targetName: "E")

        let item_CI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                                  escapeNodeIdentifier: F_IDENTIFIER,
                                  targetRssi: CF_RSSI + FI_RSSI,
                                  escapeRssi: CF_RSSI,
                                  targetName: "I")

        node.visibleNodeItems.onNext([item_CB,
                                      item_CA,
                                      item_CD,
                                      item_CE,
                                      item_CF,
                                      item_CG,
                                      item_CH,
                                      item_CI])

        return node
    }

    static func generateNode_D() -> BTNode {
```

```
    let node = BTNode(name: "D", identifier: D_IDENTIFIER)

    let item_DA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                              escapeNodeIdentifier: D_IDENTIFIER,
                              targetRssi: DA_RSSI,
                              escapeRssi: 0,
                              targetName: "A")

    let item_DC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                              escapeNodeIdentifier: D_IDENTIFIER,
                              targetRssi: DC_RSSI,
                              escapeRssi: 0,
                              targetName: "C")

    let item_DE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                              escapeNodeIdentifier: D_IDENTIFIER,
                              targetRssi: DE_RSSI,
                              escapeRssi: 0,
                              targetName: "E")

    let item_DG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                              escapeNodeIdentifier: D_IDENTIFIER,
                              targetRssi: DG_RSSI,
                              escapeRssi: 0,
                              targetName: "G")

    let item_DB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                              escapeNodeIdentifier: C_IDENTIFIER,
                              targetRssi: DC_RSSI + CB_RSSI,
                              escapeRssi: DC_RSSI,
                              targetName: "B")

    let item_DF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                              escapeNodeIdentifier: C_IDENTIFIER,
                              targetRssi: DC_RSSI + CF_RSSI,
                              escapeRssi: DC_RSSI,
                              targetName: "F")

    let item_DH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                              escapeNodeIdentifier: C_IDENTIFIER,
                              targetRssi: DC_RSSI + CH_RSSI,
                              escapeRssi: DC_RSSI,
                              targetName: "H")

    let item_DI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                              escapeNodeIdentifier: G_IDENTIFIER,
                              targetRssi: DG_RSSI + GI_RSSI,
                              escapeRssi: DG_RSSI,
                              targetName: "I")

    node.visibleNodeItems.onNext([item_DA,
                                  item_DC,
                                  item_DE,
                                  item_DG,
                                  item_DB,
                                  item_DF,
                                  item_DH,
                                  item_DI])

    return node
}

static func generateNode_E() -> BTNode {
    let node = BTNode(name: "E", identifier: E_IDENTIFIER)

    let item_EA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                              escapeNodeIdentifier: E_IDENTIFIER,
                              targetRssi: EA_RSSI,
                              escapeRssi: 0,
                              targetName: "A")
```

```
    let item_ED = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                              escapeNodeIdentifier: E_IDENTIFIER,
                              targetRssi: ED_RSSI,
                              escapeRssi: 0,
                              targetName: "D")

    let item_EG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                              escapeNodeIdentifier: E_IDENTIFIER,
                              targetRssi: EG_RSSI,
                              escapeRssi: 0,
                              targetName: "G")

    let item_EC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                              escapeNodeIdentifier: A_IDENTIFIER,
                              targetRssi: EA_RSSI + AC_RSSI,
                              escapeRssi: EA_RSSI,
                              targetName: "C")

    let item_EB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                              escapeNodeIdentifier: A_IDENTIFIER,
                              targetRssi: EA_RSSI + AB_RSSI,
                              escapeRssi: EA_RSSI,
                              targetName: "B")

    let item_EF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                              escapeNodeIdentifier: A_IDENTIFIER,
                              targetRssi: EA_RSSI + AC_RSSI + CF_RSSI,
                              escapeRssi: EA_RSSI,
                              targetName: "F")

    let item_EH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                              escapeNodeIdentifier: G_IDENTIFIER,
                              targetRssi: EG_RSSI + GH_RSSI,
                              escapeRssi: EG_RSSI,
                              targetName: "H")

    let item_EI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                              escapeNodeIdentifier: G_IDENTIFIER,
                              targetRssi: EG_RSSI + GI_RSSI,
                              escapeRssi: EG_RSSI,
                              targetName: "I")

    node.visibleNodeItems.onNext([item_EA,
                                  item_ED,
                                  item_EG,
                                  item_EC,
                                  item_EB,
                                  item_EF,
                                  item_EH,
                                  item_EI])

    return node
}

static func generateNode_F() -> BTNode {
    let node = BTNode(name: "F", identifier: F_IDENTIFIER)

    let item_FB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                              escapeNodeIdentifier: F_IDENTIFIER,
                              targetRssi: FB_RSSI,
                              escapeRssi: 0,
                              targetName: "B")

    let item_FC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                              escapeNodeIdentifier: F_IDENTIFIER,
                              targetRssi: FC_RSSI,
                              escapeRssi: 0,
                              targetName: "C")
```

```
        let item_FH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                                  escapeNodeIdentifier: F_IDENTIFIER,
                                  targetRssi: FH_RSSI,
                                  escapeRssi: 0,
                                  targetName: "H")

        let item_FI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                                  escapeNodeIdentifier: F_IDENTIFIER,
                                  targetRssi: FI_RSSI,
                                  escapeRssi: 0,
                                  targetName: "I")

        let item_FA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: FC_RSSI + CA_RSSI,
                                  escapeRssi: FC_RSSI,
                                  targetName: "A")

        let item_FD = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: FC_RSSI + CD_RSSI,
                                  escapeRssi: FC_RSSI,
                                  targetName: "D")

        let item_FG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: FC_RSSI + CG_RSSI,
                                  escapeRssi: FC_RSSI,
                                  targetName: "G")

        let item_FE = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                  escapeNodeIdentifier: C_IDENTIFIER,
                                  targetRssi: FC_RSSI + CA_RSSI + AE_RSSI,
                                  escapeRssi: FC_RSSI,
                                  targetName: "E")

        node.visibleNodeItems.onNext([item_FB,
                                      item_FC,
                                      item_FH,
                                      item_FI,
                                      item_FA,
                                      item_FD,
                                      item_FG,
                                      item_FE])

        return node
    }

    static func generateNode_G() -> BTNode {
        let node = BTNode(name: "G", identifier: G_IDENTIFIER)

        let item_GC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                                  escapeNodeIdentifier: G_IDENTIFIER,
                                  targetRssi: GC_RSSI,
                                  escapeRssi: 0,
                                  targetName: "C")

        let item_GD = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                  escapeNodeIdentifier: G_IDENTIFIER,
                                  targetRssi: GD_RSSI,
                                  escapeRssi: 0,
                                  targetName: "D")

        let item_GE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                                  escapeNodeIdentifier: G_IDENTIFIER,
                                  targetRssi: GE_RSSI,
                                  escapeRssi: 0,
                                  targetName: "E")

        let item_GH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
```

```
                                escapeNodeIdentifier: G_IDENTIFIER,
                                targetRssi: GH_RSSI,
                                escapeRssi: 0,
                                targetName: "H")

        let item_GI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                                escapeNodeIdentifier: G_IDENTIFIER,
                                targetRssi: GI_RSSI,
                                escapeRssi: 0,
                                targetName: "I")

        let item_GB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                                escapeNodeIdentifier: C_IDENTIFIER,
                                targetRssi: GC_RSSI + CB_RSSI,
                                escapeRssi: GC_RSSI,
                                targetName: "B")

        let item_GA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                                escapeNodeIdentifier: E_IDENTIFIER,
                                targetRssi: GE_RSSI + EA_RSSI,
                                escapeRssi: GE_RSSI,
                                targetName: "A")

        let item_GF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                escapeNodeIdentifier: C_IDENTIFIER,
                                targetRssi: GC_RSSI + CF_RSSI,
                                escapeRssi: GC_RSSI,
                                targetName: "F")

        node.visibleNodeItems.onNext([item_GC,
                                      item_GD,
                                      item_GE,
                                      item_GH,
                                      item_GI,
                                      item_GB,
                                      item_GA,
                                      item_GF])

        return node
    }

    static func generateNode_H() -> BTNode {
        let node = BTNode(name: "H", identifier: H_IDENTIFIER)

        let item_HC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                                escapeNodeIdentifier: H_IDENTIFIER,
                                targetRssi: HC_RSSI,
                                escapeRssi: 0,
                                targetName: "C")

        let item_HG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                                escapeNodeIdentifier: H_IDENTIFIER,
                                targetRssi: HG_RSSI,
                                escapeRssi: 0,
                                targetName: "G")

        let item_HF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                escapeNodeIdentifier: H_IDENTIFIER,
                                targetRssi: HF_RSSI,
                                escapeRssi: 0,
                                targetName: "F")

        let item_HI = BTRouteItem(targetNodeIdentifier: I_IDENTIFIER,
                                escapeNodeIdentifier: H_IDENTIFIER,
                                targetRssi: HI_RSSI,
                                escapeRssi: 0,
                                targetName: "I")

        let item_HD = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                escapeNodeIdentifier: C_IDENTIFIER,
```

```
                                targetRssi: HC_RSSI + CD_RSSI,
                                escapeRssi: HC_RSSI,
                                targetName: "D")

        let item_HE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                                escapeNodeIdentifier: G_IDENTIFIER,
                                targetRssi: HG_RSSI + GE_RSSI,
                                escapeRssi: HG_RSSI,
                                targetName: "E")

        let item_HB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                                escapeNodeIdentifier: F_IDENTIFIER,
                                targetRssi: HF_RSSI + FB_RSSI,
                                escapeRssi: HF_RSSI,
                                targetName: "B")

        let item_HA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                                escapeNodeIdentifier: C_IDENTIFIER,
                                targetRssi: HC_RSSI + CA_RSSI,
                                escapeRssi: HC_RSSI,
                                targetName: "A")

        node.visibleNodeItems.onNext([item_HC,
                                      item_HG,
                                      item_HF,
                                      item_HI,
                                      item_HD,
                                      item_HE,
                                      item_HB,
                                      item_HA])

        return node
    }

    static func generateNode_I() -> BTNode {
        let node = BTNode(name: "I", identifier: I_IDENTIFIER)

        let item_IH = BTRouteItem(targetNodeIdentifier: H_IDENTIFIER,
                                escapeNodeIdentifier: I_IDENTIFIER,
                                targetRssi: IH_RSSI,
                                escapeRssi: 0,
                                targetName: "H")

        let item_IG = BTRouteItem(targetNodeIdentifier: G_IDENTIFIER,
                                escapeNodeIdentifier: I_IDENTIFIER,
                                targetRssi: IG_RSSI,
                                escapeRssi: 0,
                                targetName: "G")

        let item_IF = BTRouteItem(targetNodeIdentifier: F_IDENTIFIER,
                                escapeNodeIdentifier: I_IDENTIFIER,
                                targetRssi: IF_RSSI,
                                escapeRssi: 0,
                                targetName: "F")

        let item_IC = BTRouteItem(targetNodeIdentifier: C_IDENTIFIER,
                                escapeNodeIdentifier: F_IDENTIFIER,
                                targetRssi: FI_RSSI + FC_RSSI,
                                escapeRssi: FI_RSSI,
                                targetName: "C")

        let item_ID = BTRouteItem(targetNodeIdentifier: D_IDENTIFIER,
                                escapeNodeIdentifier: G_IDENTIFIER,
                                targetRssi: IG_RSSI + GD_RSSI,
                                escapeRssi: IG_RSSI,
                                targetName: "D")

        let item_IE = BTRouteItem(targetNodeIdentifier: E_IDENTIFIER,
                                escapeNodeIdentifier: G_IDENTIFIER,
                                targetRssi: IG_RSSI + GE_RSSI,
```

```
                                escapeRssi: IG_RSSI,
                                targetName: "E")

        let item_IB = BTRouteItem(targetNodeIdentifier: B_IDENTIFIER,
                                escapeNodeIdentifier: F_IDENTIFIER,
                                targetRssi: FI_RSSI + FB_RSSI,
                                escapeRssi: FI_RSSI,
                                targetName: "B")

        let item_IA = BTRouteItem(targetNodeIdentifier: A_IDENTIFIER,
                                escapeNodeIdentifier: F_IDENTIFIER,
                                targetRssi: FI_RSSI + FB_RSSI + BA_RSSI,
                                escapeRssi: FI_RSSI,
                                targetName: "A")

        node.visibleNodeItems.onNext([item_IH,
                                      item_IG,
                                      item_IF,
                                      item_IC,
                                      item_ID,
                                      item_IE,
                                      item_IB,
                                      item_IA])

        return node
    }
}
```