

ТЕХНИЧЕСКИ УНИВЕРСИТЕТ ВАРНА
Факултет по Изчислителна Техника и Автоматизация
Катедра „Компютърни науки и технологии“

**„Реализация на система за симулиране на
Bluetooth Low Energy (BLE) технология“**

**Дипломна работа за придобиване на
образователно-квалификационна степен „бакалавър“**

Дипломант:

Йордан Велков Йорданов

Факултетен № 17621320

Специалност: Компютърни Системи и Технологии

Научен ръководител:

гл. ас. д-р инж. А. Хъкъ

Варна 2021

Съдържание

Глава 1. Увод.....	4
Глава 2. Bluetooth Low Energy (BLE) технология – стандарт, принцип на работа, предимства и недостатъци	6
2.1 Bluetooth Low Energy стандарт	6
2.1.1 Архитектура на BLE	6
2.1.2 Physical Layer (PHY)	8
2.1.3 Link Layer	10
2.1.4 Bluetooth адрес.....	11
2.2 Принцип на работа.....	12
2.2.1 Рекламирање (Advertising)	12
2.2.2 Свързване.....	14
2.2.3 Изпращане на даннови пакети	16
2.3 Формат на пакетите	17
2.3.1 Пакет с PDU ADV_IND	18
2.3.2 Пакет с PDU ADV_NONCONN_IND.....	18
2.3.3 Пакет с PDU CONNECT_IND или AUX_CONNECT_REQ.....	19
2.3.4 Пакет с PDU ADV_EXT_IND	20
2.3.5 Пакет с PDU AUX_ADV_IND	22
2.3.6 Пакет с PDU AUX_CONNECT_RSP	22
2.3.7 Пакети с LL Data PDU	22
2.3.8 Пакети с LL Control PDU.....	23
2.4 Предимства и недостатъци на Bluetooth Low Energy спрямо други сходни технологии.....	24
2.4.1 Предимства.....	24
2.4.2 Недостатъци.....	25
Глава 3. Съществуващи решения	27
3.1 BLE Peripheral Simulator.....	27
3.2 Bluetooth Low Energy Communication интерфейса на MATLAB.....	28
Глава 4. Избор на среда за създаване на симулатор на BLE технология	30
4.1 Спецификация на изискванията към приложението и цели	30
4.2 Използвани технологии	31
4.2.1 Десктоп приложение	31
4.2.2 Java	31

4.2.3 Apache Maven	32
4.2.4 JavaFX & Scene Builder	33
4.2.4.1 Характеристики на JavaFX	34
4.2.4.2 Scene Builder	35
4.2.5 Launch4j	35
Глава 5. Структура на създавания симулатор	37
5.1 Архитектура на директориите	37
5.2 Потребителски интерфейс	39
5.2.1 Главен прозорец	39
5.2.1.1 Табът Scene	39
5.2.1.2 Табът Packet Sniffer (Master)	40
5.2.1.3 Табът Message Diagrams	41
5.2.1.4 Табът Device Statistics	41
5.2.1.5 Добавяне на ново устройство	42
5.2.1.6 Свързване на периферно устройство с главното	43
5.2.1.7 Изпращане на даннови пакети от подчиненото към главното устройство	43
5.2.1.8 Информация за пакетите	44
5.3 Работа на приложението	44
Глава 6. Тестване на симулатора	54
Глава 7. Заключение и изводи	60
Източници	61
Приложение 1. Терминологичен речник	62
Приложение 2. Ръководство на потребителя	64
Приложение 3. Код на приложението	70

Глава 1. Увод

Употребата на безжични устройства в нашето ежедневие бързо се разраства с всеки ден. Една от поставените цели на различни компании и изследователи, още от зората на радио технологиите е да се създадат най-ефективните, успешни и евтини за производство модули, които да бъдат пуснати на пазара.

Функцията на различните модули е в зависимост от нуждите, поради които са създадени. В някои ситуации трябва да се изпратят големи по обем данни чрез безжична връзка, в други трябва да се оптимизира консумацията на енергия, или използване на устройство, което уведомява за появата на друго устройство.

Bluetooth е един от най-популярните безжични протоколи и е наличен в смартфони, компютри и други устройства. Експлозивният ръст на Bluetooth устройствата и новите случаи на употреба доведоха Bluetooth SIG и други компании до осъзнаването, че Bluetooth консумира твърде много енергия и отнема твърде много време за свързване в някои приложения. Когато става дума за енергийно ефективни радио модули, които могат да изпращат подходящо количество данни при нисък разход на енергия, една от технологиите които се откроява е Bluetooth Low Energy (BLE).

Технологията BLE има широк спектър на приложение в различни области като: медицина, в която може да има сензори за измерване на глюкоза или на кръвно налягане, и той да докладва статуса си като използва BLE технологията. В образованието, където може да се направи система, която автоматично да следи присъствието на ученици, които носят със себе си идентифициращо устройство. Друга сфера е селското стопанство, където може да използваме beacons, за събиране и анализиране на данни за околната среда, като температура на въздуха, влажност на почвата и други сензорни измервания, които да помогнат за ефективно и устойчиво производство на храни. Не на последно място е приложение в домашната автоматизация, където възможните реализации множество.

Днес почти всички смартфони и таблети са съвместими с BLE, което означава, че могат безпроблемно да комуникират с множество Bluetooth устройства, като Bluetooth слушалки, фитнес тракери, смарт часовници и много други, като целта е подобряване и улесняване на ежедневието.

Целта на настоящата разработка е да създаде симулатор за BLE с отворен код, който позволява симулиране на свързаност между устройства в BLE мрежа, изследване на основната функционалност, базирана на стандарта, както и представяне на обменните съобщения между Master и Slave устройства при реализиране процеса на свързване, изпращане на данни и прекратяване на връзката.

Задачи за постигане на целта:

1. Запознаване с технологията BLE;
2. Запознаване с модела, предоставян от BLE стандарти за общо ползване;
3. Запознаване с процеса на свързване, изпращане на данни и прекратяване на връзката между Master и Slave;
4. Реализация и тестване.

Глава 2. Bluetooth Low Energy (BLE) технология – стандарт, принцип на работа, предимства и недостатъци

2.1 Bluetooth Low Energy стандарт

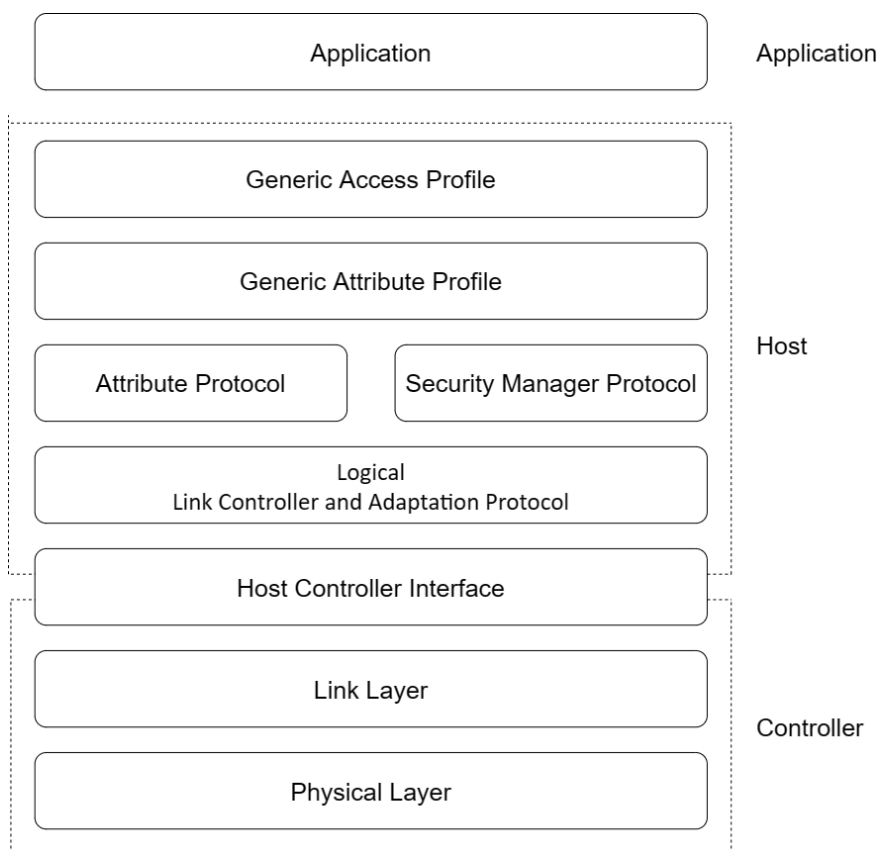
Bluetooth стартира като технология за подмяна на кабелни устройства на близки разстояния, като мишка, клавиатура или слушалки. Първата официална версия на Bluetooth е публикувана през 1994 от Ericsson. Понастоящем има 2 типа Bluetooth устройства – едните са познати като Bluetooth Classic (Basic Rate / Enhanced Data Rate) и се използват в безжични високоговорители, автомобилни системи и др., а втория вид е Bluetooth Low Energy (BLE).

Технологията BLE е въведена в Bluetooth стандарта във версия 4.0 и е по известна в приложения, при които консумацията на енергия е от решаващо значение и данните за пренос са с малки количества.

Тези два типа устройства са несъвместими помежду си, въпреки че споделят една и съща марка и спецификация. Устройство с Bluetooth Classic не може да комуникира директно с BLE устройство. Ето защо някои устройство като смартфони внедряват и двата типа (наричани още Dual Mode Bluetooth устройства), като това позволява да могат да комуникират и с двата типа Bluetooth [1].

2.1.1 Архитектура на BLE

Архитектурата на BLE (Фиг. 1) включва различни компоненти [2]. При тази архитектура физически и Link Layer слоевете се групират в структура наречена контролер, а останалите слоеве се групират в структура хост. Интерфейсът хост към контролер (HCI – Host-Controller Interface) стандартизира комуникацията между контролера и хоста.



Фиг. 1. Архитектура на Bluetooth Low Energy

Структура контролер:

- Физически слой (Physical Layer): управлява радиопредаването / приемането;
- Канален Слой (Link Layer): определя структурата на пакетите, включва машината на състоянията (state machine) и радио управлението, и осигурява криптиране на това ниво.

Структура хост:

- L2CAP (Logical Link Control and Adaptation Layer Protocol): протокол за управление и адаптация на логическа връзка. L2CAP действа като протоколен мултиплексор и обработва сегментирането и повторното сглобяване на пакети. Той също така предоставя логически канали, които се мултиплексират по една или повече логически връзки. L2CAP, използван в BLE, е оптимизиран и опростен протокол, базиран на класическия Bluetooth L2CAP;

- **ATT (Attribute Protocol):** протоколът за атрибути осигурява средства за предаване на данни между BLE устройства. Разчита на BLE връзка и осигурява процедури за четене, запис, индикация и уведомяване на стойностите на атрибутите по тази връзка. ATT се използва в повечето BLE приложения и понякога в BR / EDR приложения;

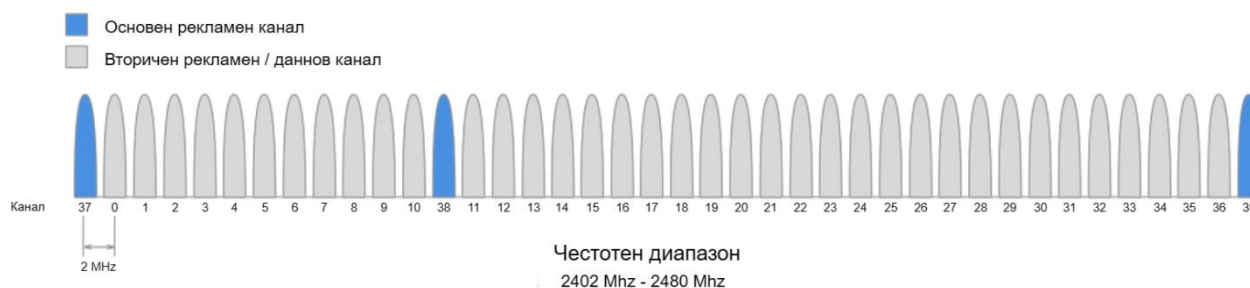
- **GATT (General Attribute Profile):** общ профил на атрибутите. GATT се използва за групиране на отделни атрибути в логически услуги, например услуга сърдечна честота, която излага работата на сензор за сърдечен ритъм. В допълнение към действителните данни, GATT предоставя и информация за атрибутите, т.е. как могат да бъдат достъпни и какво ниво на сигурност е необходимо;

- **GAP (General Access Profile):** общ профил за достъп. GAP слойт осигурява средства за BLE устройства да рекламират себе си или други устройства, правят откриване на устройства, отварят и управляват връзки и излъчват данни;

- **SM (Security Manager):** мениджър на сигурността. Осигурява средства за свързване на устройства, криптиране и декриптиране на данни и активиране на поверителността на устройствата.

2.1.2 Physical Layer (PHY)

Физическият слой (PHY) се отнася до радио хардуера, използван за комуникация и за модулация / демодулация на данните. BLE работи в ISM обхвата (2,4 GHz спектър), който е сегментиран в 40 радио-честотни канала, всеки раздалечен през 2 MHz, както е показано на следната Фигура:



Фиг. 2. Честотен диапазон и индексирание на каналите

Три от тези канали се наричат първични рекламни канали (primary advertising channels), докато останалите 37 се използват като вторични рекламни канали (secondary advertising channels) и за пренос на данни по време на връзка.

Рекламата винаги започва с изпращането на рекламните пакети на трите канала предназначени за това (или подгрупа от тези канали). Това позволява на устройствата да сканират рекламодатели, за да ги намерят и да прочетат техните рекламни данни. След това сканиращото устройство може да инициира връзка, ако рекламодателят го позволява. Той може също така да поиска това, което се нарича scan request, и ако рекламодателят поддържа тази функционалност, той ще отговори със scan response. Scan Request и Scan Response позволяват на рекламодателя да изпраща допълнителни рекламни данни на устройства, които се интересуват от получаването им.

На физическото ниво се използва спектър за честотно прескачане (Frequency Hopping Spread Spectrum - FHSS), който позволява и двете комуникационни устройства да превключват на произволно съгласувано избрани честоти за обмен на данни. Това значително подобрява надеждността и позволява на устройствата да избягват честотни канали, които могат да бъдат претоварени и използвани от други устройства в заобикаляща среда.

Технологията Bluetooth позволява свързване на устройства с ниски мощности. Нивата на мощност на предаване са:

- Максимално: 100mW (+20 dBm) за версии 5 и нагоре, 10mW (+10 dBm) за версии 4.2 и надолу
- Минимално: 0,01 mW (-20 dBm)

В по-старите версии на Bluetooth (4.0, 4.1 и 4.2) скоростта на предаване на данни е фиксирана на 1 Mbps. Радиото на физически слой (PHY) в този случай се нарича 1M PHY и е задължително във всички версии, включително Bluetooth 5. С Bluetooth 5 обаче, са въведени две нови опционални PHY:

- 2Mbps PHY (2M PHY), използван за постигане на двойна скорост от по-ранните версии на Bluetooth;
- Кодиран PHY (CODED PHY), използван за комуникация на по-голям обхват.

2.1.3 Link Layer

Link Layer е слой, който взаимодейства с физическия слой и осигурява на слоевете от по-високо ниво абстракция и начин за взаимодействие с радиото (чрез междинно ниво, наречено HCI слой). Той е отговорен за управление на състоянието на радиото, както и изискванията за синхронизация, необходими за задоволяване BLE спецификацията. Също така отговаря за управлението на хардуерни операции като CRC, генериране на случайни числа и криптиране.

Слойт Link Layer управлява различните състояния на радиото (Фиг. 3). Трите основни състояния, в които работи BLE устройство, са:

- Състояние на рекламата;
- Състояние на сканиране;
- Свързано състояние.



Фиг. 3. Състояния на свързващия слой

Когато дадено устройство рекламира, то позволява на други устройства, които сканират, да намерят устройството и евентуално да се свържат с него. Ако рекламното устройство позволява връзки и сканиращо устройство го намери и реши да се свърже с него, и двете устройства влизат в свързано състояние.

- В готовност: състояние по подразбиране, при което радиото не предава и не приема никакви данни;
- Рекламирање: състоянието, в което устройството изпраща рекламни пакети на други устройства за откриване и четене;
- Сканиране: състоянието, в което устройството сканира за устройства, които са рекламиращи;
- Инициране: състоянието, в което сканиращото устройство решава да установи връзка с устройство, което рекламира;
- Свързан: състоянието, в което дадено устройство има установена връзка с друго устройство и редовно обменя данни с него. Това се отнася както за устройството, което е било в състоянието на реклама или такова, което сканира за реклами и след това реши да го направи иницира връзка с рекламиращото устройство. В това свързано състояние устройството, което иницира връзката се нарича master, а устройството, което е рекламирало - slave.

2.1.4 Bluetooth адрес

Bluetooth устройствата се идентифицират с 48-битов адрес [1], подобен на MAC адрес. Има два основни типа адреси: публични адреси и случайни адреси.

Публичен адрес: това е фиксиран адрес, който не се променя и е фабрично програмиран. Трябва да е регистриран в IEEE (подобно на MAC адрес на WiFi или Ethernet устройство).

Случаен адрес: тъй като производителите имат избор какъв тип адрес да използват (случаен или публичен), случайните адреси са по-популярни, защото не изискват регистрация в IEEE. Случаен адрес се програмира на устройството или се генерира по време на изпълнение. Може да е един от двата подтипа:

- Статичен адрес
 - Използва се като заместител на публични адреси;
 - Може да се генерира при зареждане или да остане същият през целия цикъл на живот;
 - Не може да се промени до цикъл на захранване.

- Частен адрес

Този също е разделен на два допълнителни подтипа:

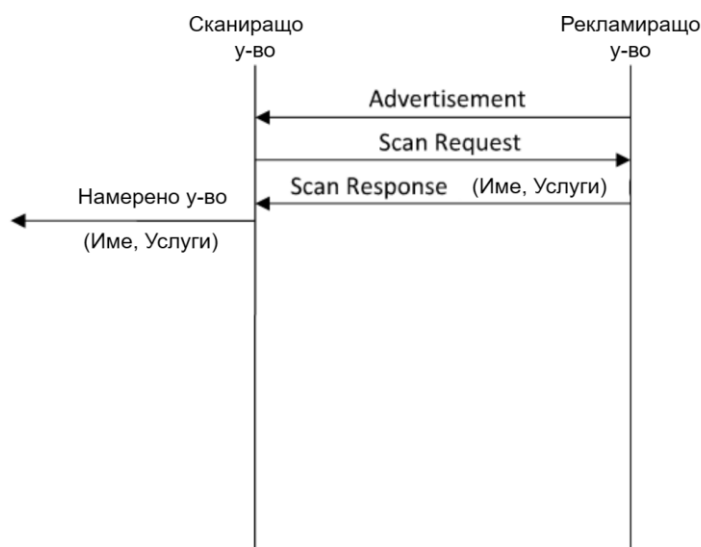
- Неразрешим частен адрес:
 - Случаен, за определено време;
 - Не се използва често.
- Разрешим частен адрес:
 - Използва се за поверителност;
 - Генериран с помощта на ключ за разрешаване на идентичност (Identity Resolving Key) и произволно число;
 - Променя се периодично (дори по време на живота на връзката);
 - Използва се, за да се избегне проследяване от неизвестни скенери;
 - Надеждните устройства може да го разрешат с помощта на предварително съхранения IRK.

2.2 Принцип на работа

2.2.1 Рекламирање (Advertising)

Рекламата е една от най-фундаменталните операции в BLE безжичната технология. Рекламата дава начин на устройства, да излъчват присъствието си, като така позволява да се установяват връзки и опционално да се изпращат данни като списъка на поддържаните услуги или името на устройството и други.

На Фиг. 4 се илюстрира BLE устройство, което рекламира, излъчва пакети на един или няколко рекламни канала, които отдалечените устройства след това могат да прихванат.



Фиг. 4. Събитие на активно сканиране

Приложението обикновено има контрол върху рекламните параметри представени в таблица 1.

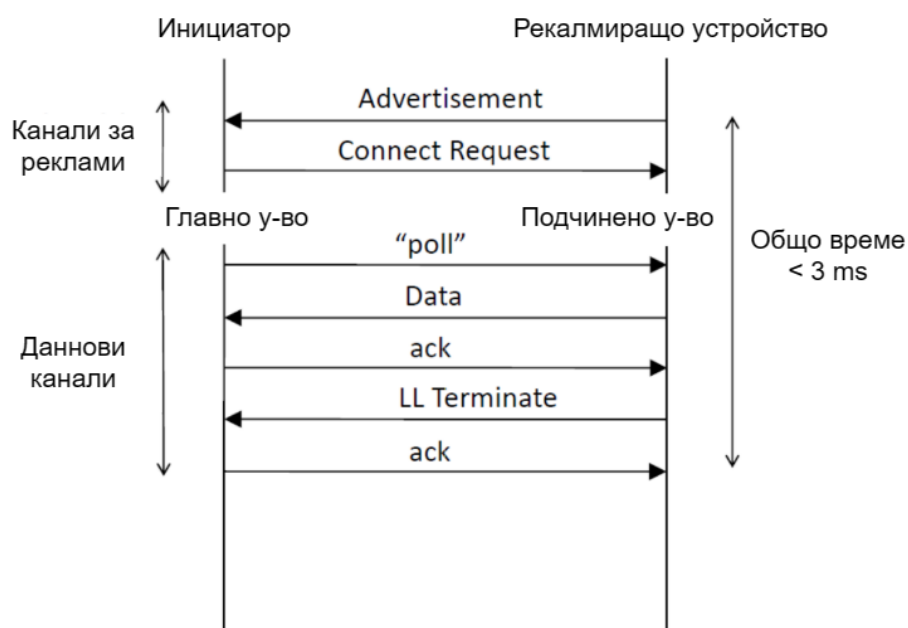
Таблица. 1. Рекламни параметри

Параметър	Стойности	Описание
Интервал на реклама (Advertising Interval)	20 ms - 10240 ms	Определя интервала между рекламната събития. Всяко събитие се състои от 1 до 3 реклами пакети в зависимост от конфигурацията. Случайна стойност от 0-10 ms се добавя от свързващия слой към всеки рекламен интервал, за да се избегне сблъсък на пакети
Рекламни канали (Advertisement channels)	37, 38 и 39 (основни рекламни канали) 0 - 36 (даннови канали)	Физическите радиочестотни канали, използвани за предаване на рекламните пакети. За най-надеждна работа трябва да се използват всички канали, но намалявайки броя на използваните канали ще намалят консумацията на енергия при разходи за надеждност.
Режим на откриваемост (Discoverability mode)	Неоткриваем; Общо откриваем; Ограничено откриваем, Излъчване (Broadcasting)	Определя как рекламиращото устройство е видим за други устройства
Режим за свързване (Connectability mode)	Не може да се свърже; Директно свързваем; Ненасочено свързваем	Определя дали рекламиращото устройство може да бъде свързано или не
Данни (Payload)	0 до 31 В (първична реклама); 0 до 255 В (BT5 вторична реклама)	Във всеки рекламен пакет на основните канали може да се включат 0-31 байта данни. Във всеки вторичен рекламен пакет могат да бъдат включени 0-255 байта данни (Bluetooth 5)

2.2.2 Свързване

Връзките позволяват предаването на данни от приложенията по надежден начин, тъй като при BLE те използват CRC (Cyclic Redundancy Check), потвърждения (acknowledgements) и препредаване на изгубени данни, за да се осигури коректно им доставяне. В допълнение, BLE връзките използват Adaptive Frequency Hopping (AFH) за откриване и адаптиране към околните радиочестотни условия и осигуряване на надежден физически слой. Връзките също поддържат криптиране и декриптиране на данни, за да се гарантира тяхната поверителност.

BLE връзките винаги се инициират от сканиращото устройство, което получава рекламен пакет, включващ факта, че устройството, което рекламира позволява връзки. На Фиг. 5 е илюстрирано как се осъществява BLE връзка.



Фиг. 5. Събитие, при което има установяване на връзка, предаване на един пакет и прекратяване на връзката

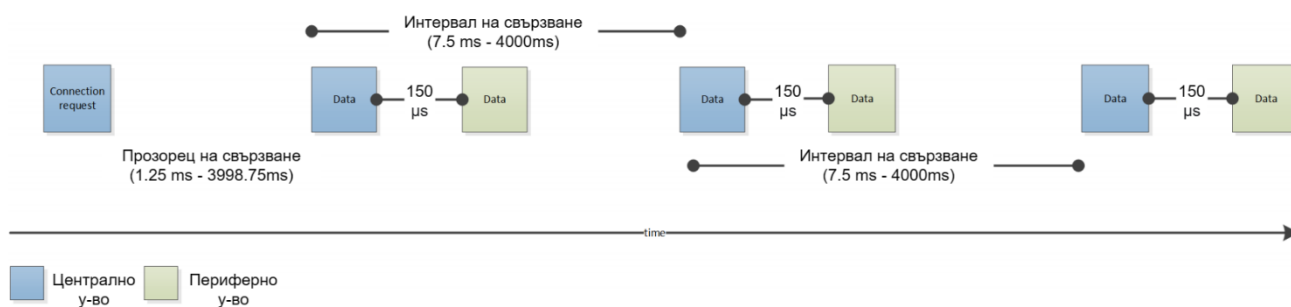
Приложението обикновено контролира параметрите на свързване, представени в таблица 2.

Таблица. 2. Параметри на свързване

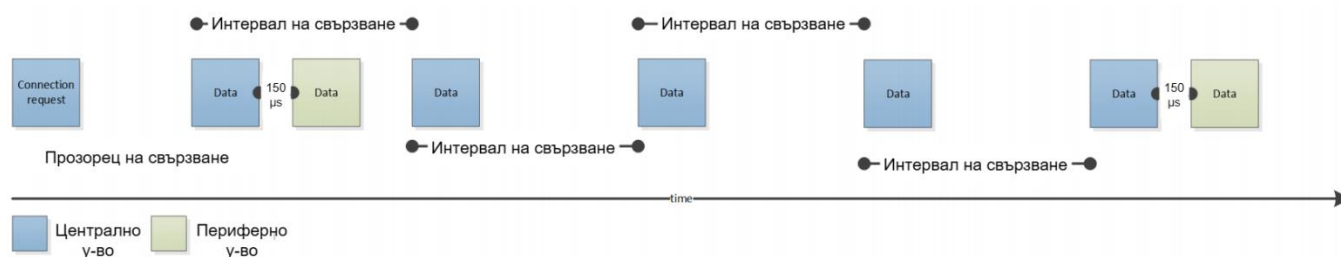
Параметър	Стойности	Описание
Минимален интервал на връзка (Minimum Connection Interval)	7.5 ms	Минимално разрешен интервал на свързване
Максимален интервал на връзка (Maximum Connection Interval)	4000 ms	Максимално разрешен интервал на свързване
Латентност на връзката (Latency)	0 to 500 (интервала)	Броят на събитията за свързване, на което периферното устройство е разрешено да пропусне, ако няма данни за изпращане
Време за изчакване на надзора (Supervision timeout)	100 ms to 32000 ms	Определя колко дълго може да бъде прекъсването в комуникациите (например поради ситуация извън обхвата) преди връзката да отпадне и да се покаже грешката на потребителя.

Параметрите на връзката могат да бъдат актуализирани по време на живота ѝ, като се използва съобщение за актуализация на връзката.

Събитието за връзка (Фиг. 6) започва, когато централното устройство изпрати пакет до периферното устройство на определения интервал на свързване. Периферното устройство може да реагира на 150 μ s, след като е получило пакет от централното устройство. Ако периферното устройство няма данни за изпращане, то може да пропусне определен брой събития на връзката, определени от параметъра на периферната латентност (Фиг. 7). Ако централното или периферното устройство не получи пакети в рамките на времето, определено от таймаут на надзора, връзката се прекъсва.



Фиг. 6 Хронология на връзката



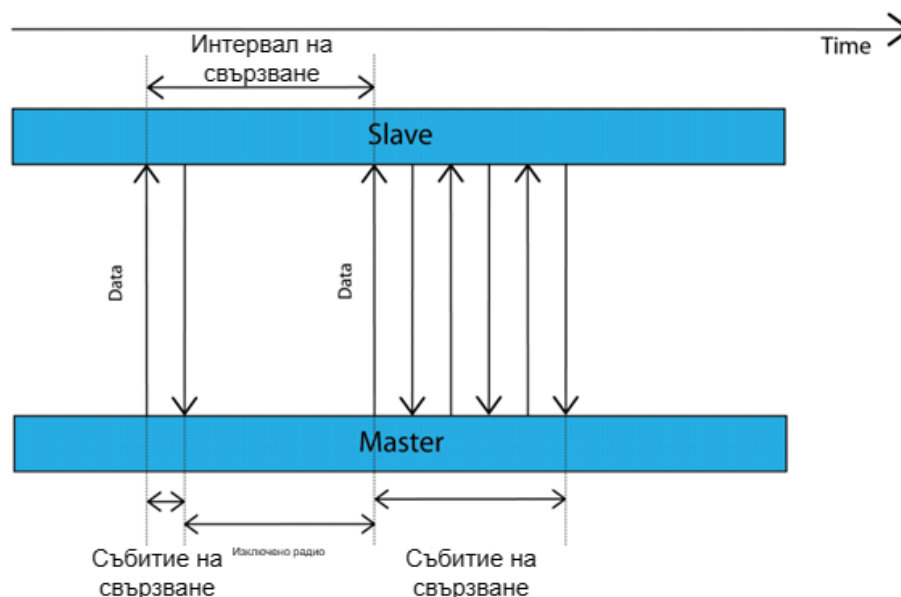
Фиг. 7. Латентност на периферното устройство (при латентност = 3)

Ако периферното устройство има повече данни за изпращане, отколкото могат да бъдат вградени в един пакет, събитието за връзка автоматично ще се удължи и периферното устройство може да изпрати толкова пакети, колкото е времето до началото на следващия интервал на свързване. Това може да се използва само при операции с протокол за приписване, които не изискват потвърждение [2].

2.2.3 Изпращане на даннови пакети

Веднъж свързани, главното и подчиненото устройство обменят даннови пакети на редовни интервали, наречени „събития на връзката“.

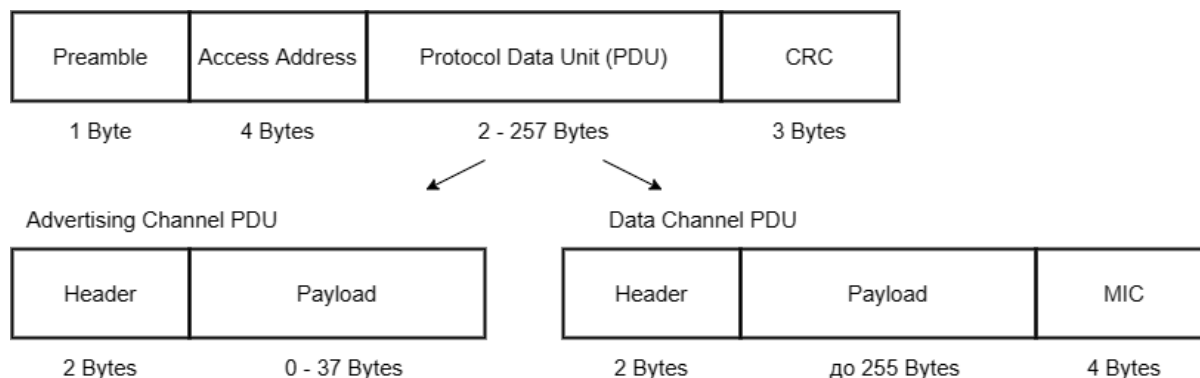
По време на така нареченото събитие за връзка, устройствата алтернативно изпращат даннови пакети един към друг, докато нито една от страните не остане без повече данни за изпращане. Събитието на връзка се случва периодично и непрекъснато, докато връзката се затвори или изгуби. Трябва да се съдържа поне един пакет, изпратен от главното устройство, като подчиненото винаги изпраща пакет обратно, ако получи такъв от главния. Ако главното устройство не получи обратно пакет от подчиненото, то ще затвори събитието на връзка и ще възобнови изпращането на пакети при следващото събитие. Интервалът на събитието е между 7,5 ms до 4 s (размер на стъпката: 1,25 ms) и ако няма чакащи данни за обмен, празни 0-байтови даннови пакети се обменят, за да се поддържа връзка [1].



Фиг. 8. Интервалът на връзката и събитията на връзката

2.3 Формат на пакетите

Свързващият слой на BLE има само един формат на пакетите [3], използван както за пакети от рекламни канали, така и за даннови пакети (Фиг. 9).



Фиг. 9. Формат на пакетите в BLE

Отделните полета на пакета имат следните функции:

- **Preamble:** използва се от приемника за синхронизация (по време и честота) и за извършване на Automatic Gain Control (автоматичен контрол на усилването). Това е предварително дефиниран шаблон с размер 1 байт, който е известен на приемника. Рекламния пакет използва "10101010" в двоична бройна система, а данновия използва "10101010" или "01010101" в зависимост от най-младшия бит в двоична форма;

- Адрес за достъп (Access Address): за всички рекламни пакети се използва фиксиран шаблон "0x8E89BED6" в шестнадесетична форма с размер от 4 октета (32 бита). За данновите пакети, то се състои от 32 битова произволна стойност, генерирана от BLE устройство в „начално състояние“. Същата стойност се използва в пакета „заявка за свързване (CONNECT_REQ)“;
- PDU (Protocol Data Unit): Състои се или от „PDU за рекламен канал“ или от „PDU за даннов канал“, както е дефинирано на Фиг. 9;
- CRC (Cyclic Redundancy Check): той е с размер 24 бита. Изчислява се през PDU. Използва се за откриване на грешки в пакета. CRC се изчислява, като се използва полином от формата $x^{24} + x^{10} + x^9 + x^6 + x^4 + x^3 + x + 1$.

В следващите подточки са представени типовете PDU, които са текущо реализирани в симулатора.

2.3.1 Пакет с PDU ADV_IND

PDU ADV_IND има payload, както е показано на Фиг. 10. Това PDU се използва при свързващи и сканируеми ненасочени рекламни събития.

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Фиг. 10. Формат на payload на пакет с ADV_IND PDU

Payload се състои от полетата AdvA и AdvData. Полето AdvA съдържа публичния или случаен адрес на рекламодателя, а полето AdvData, ако не е празно, съдържа рекламни данни.

2.3.2 Пакет с PDU ADV_NONCONN_IND

PDU ADV_NONCONN_IND има payload, показан на Фиг. 11. PDU на пакетът трябва да се използва в несвързани и не-сканируеми, ненасочени рекламни събития.

Payload	
AdvA (6 octets)	AdvData (0-31 octets)

Фиг. 11. Формат на payload на пакет с ADV_NONCONN_IND PDU

Полетата имат същите функции като при пакет с PDU ADV_IND.

2.3.3 Пакет с PDU CONNECT_IND или AUX_CONNECT_REQ

Пакети с PDU CONNECT_IND или AUX_CONNECT_REQ имат payload както е показано на Фиг. 12.

Payload		
InitA (6 octets)	AdvA (6 octets)	LLData (22 octets)

Фиг. 12. Формат на payload на пакет с CONNECT_IND или AUX_CONNECT_REQ PDU

Форматът на полето LLData е показан на Фиг. 13

LLData									
AA (4 octets)	CRCInit (3 octets)	WinSize (1 octet)	WinOffset (2 octets)	Interval (2 octets)	Latency (2 octets)	Timeout (2 octets)	ChM (5 octets)	Hop (5 bits)	SCA (3 bits)

Фиг. 13. Формат на полето LLData на пакет с CONNECT_IND или AUX_CONNECT_REQ PDU

Payload се състои от полета InitA, AdvA и LLData. Полето InitA съдържа публичния или произволния адрес на инициатора на връзката, полето AdvA съдържа публичния или произволния адрес на рекламодателя.

LLData се състои от 10 полета:

- Полето AA трябва да съдържа адреса на достъп (Access Address) на инициатора;
- Полето CRCInit трябва да съдържа инициализиращата стойност за изчислението на CRC за връзката. Това ще бъде произволна стойност, генерирана от свързващия слой;

- Полето WinSize указва стойността на прозореца на свързване, след която връзката се осъществява, ако обстоятелствата позволяват;
- Полето WinOffset указва стойността на отместване преди да започне събитие на свързване;
- Полето Interval указва интервала на свързване между двете устройства;
- Полето Latency указва каква е латентността на устройството;
- Полето Timeout сочи времето в милисекунди, след което ако не се получи пакет връзката отпада;
- Полето ChM трябва да съдържа картата на каналите, която показва кои канали ще се използват за пренос на данни и кои не. Всеки канал е представен с бит, позициониран според индекс на канал за данни;
- Полето Hop сочи какъв е хоп инкремента и също използвания алгоритъм за избор на канали. Той трябва да има произволна стойност в диапазона от 5 до 16;
- Полето SCA определя точността на часовника за заспиване (Sleep Clock Accuracy) на главното устройство.

2.3.4 Пакет с PDU ADV_EXT_IND

Пакет с PDU ADV_EXT_IND използва общия разширен payload формат, показан на Фиг . 14.

Payload			
Extended Header Length (6 bits)	AdvMode (2 bits)	Extended Header (0 - 63 octets)	AdvData (0 - 254 octets)

Фиг. 14. Формат на общ разширен payload

Полетата имат следните характеристики:

- Дължината на разширения хедър (Extended Header Length) има стойност между 0 и 63 и показва размера на полето „Разширен хедър“ (Extended Header);
- Полето AdvMode показва вида на рекламата, която може да бъде non-connectable, non-scannable, connectable и/или scannable;

- Полето AdvData, ако присъства, съдържа рекламни данни от рекламиращото устройство. Максималният размер на AdvData зависи от размера на разширения хедър;
- Полето Extended Header, присъства, само ако полето Extended Header Length е ненулево. Форматът на разширения хедър е показан на Фиг. 15;

Extended Header								
Extended Header Flags (1 octet)	AdvA (6 octets)	TargetA (6 octets)	CTEInfo (1 octet)	AdvData Info (ADI) (2 octets)	AuxPtr (3 octets)	SyncInfo (18 octets)	TxPower (1 octet)	ACAD (varies)

Фиг. 15. Формат на разширения хедър на общия разширен payload

Extended Header Flags е с размер 8 бита, като всеки бит отговаря за поредното поле от хедъра. Ако в битът е установена 1, то значи, че полето ще присъства, ако е 0 – не присъства.

Полетата в Extended Header имат следните характеристики:

- Адреса на рекламиращото устройство (AdvA) съдържа неговия статичен или случаен адрес;
- Полето TargetA съдържа адреса на сканиращото или инициращото устройство, към което е насочена рекламата;
- CTEInfo индикира дали пакета включва Constant Tone Extension;
- Информацията за рекламните данни (AdvDataInfo) се дели на две полета (Фиг. 16):

AdvDataInfo	
Advertising Data ID (DID) (12 bits)	Advertising Set ID (SID) (4 bits)

Фиг. 16. Формат на полето AdvDataInfo

- Идентификаторът на рекламния набор (Advertising Set ID) се задава от рекламиращото устройство, за да прави разлика между различни рекламни поредици, предавани от това устройство;
- Идентификационният номер на рекламните данни (Advertising Data ID) се задава от рекламиращото устройство, за да посочи на сканиращото устройство дали може да приеме, че данните в

AdvData са дублирани на предишните AdvData, изпратени в поранен пакет.

- AuxPtr, когато е наличен, индикира, че някои или всички рекламни данни се намират в последващ допълнителен пакет;
- Синхронизиращата информация (SyncInfo), когато е налична, индикира наличието на периодично рекламиране;
- TxPower индикира предавателната мощност на антената на устройството;
- Допълнителното поле за рекламни данни на контролера (Additional Controller Advertising Data – ACAD) съдържа данни от контролера на рекламиращото устройство или са предназначени да бъдат използвани от контролера на получателя.

2.3.5 Пакет с PDU AUX_ADV_IND

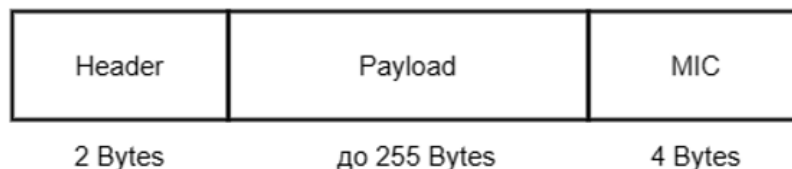
Пакет с PDU AUX_ADV_IND използва общия разширен payload формат, показан на Фиг . 14. Полетата, които ще са налични зависи от това дали рекламния пакет позволява свързване и/или сканиране.

2.3.6 Пакет с PDU AUX_CONNECT_RSP

Пакет с PDU AUX_CONNECT_RSP използва общия разширен payload формат, показан на Фиг . 14, като полетата които ще се използват в Extended Header ще са AdvA и TargetA.

2.3.7 Пакети с LL Data PDU

PDU на данновия физически канал се състои от 16 битов хедър, payload с променлива дължина и 32 битовото поле за проверка на целостта на съобщението (Message Integrity Check – MIC), което е опционално.



Фиг. 17. PDU на даннов пакет

Полето Header на PDU на физическия канал за данни е показано на Фиг. 18.

Header							
LLID (2 bits)	NESN (1 bit)	SN (1 bit)	MD (1 bit)	CP (1 bit)	RFU (2 bits)	Length (8 bits)	CTEInfo (8 bits)

Фиг. 18. Хедър на PDU на даннов пакет

Полетата в хедъра на Фиг. 18 имат следните функционалности:

Таблица 3. Описание на полетата на хедъра на PDU на даннов пакет

Поле	Описание
LLID	Индикира дали пакета е даннов или контролен
NESN	Следващ очакван номер на поредица (Next Expected Sequence Number)
SN	Номер на поредицата (Sequence Number)
MD	Повече данни (More Data)
CP	Указва дали CTEInfo присъства
Length	Размера в октети на payload-а и MIC, ако е наличен
CTEInfo	Указва типа и дължината на CTE

PDU за LL_DATA с полето LLID в хедъра, зададено на 1, и полето за дължина, зададено на 0, е известно като празен PDU. Главното устройство може да изпрати празен PDU на подчиненото, за да позволи на подчинения да отговори, включително и с празен PDU, с цел поддържане на връзката.

2.3.8 Пакети с LL Control PDU

PDU за LL (Link Layer) контрол на физически канал за данни, се използва за управление на връзката между две устройство. Payload на LL Control PDU е показан на Фиг. 19.

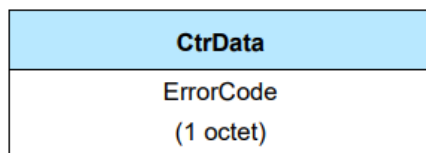
Payload	
Opcode (1 octet)	CtrData (0 – 250 octets)

Фиг. 19. Payload на контролен даннов пакет

Полето Opcode индикира типа на контролния пакет.

Полето CtrData в LL Control PDU се определя от полето Opcode. За всеки даден Opcode дължината на полето CtrData е фиксирана.

В симулатора текущо е дефиниран само един контролен пакет – LL_TERMINATE_IND. Неговия CtrData се състои от само едно поле, както е показано на Фиг. 20. Полето ErrorCode трябва да информира отдалеченото устройство защо връзката е на път да бъде прекратена.



Фиг. 20. CtrData на контролния пакет LL_TERMINATE_IND

2.4 Предимства и недостатъци на Bluetooth Low Energy спрямо други сходни технологии

BLE технологията, която е част от най-новата спецификация за Bluetooth 4.1 от Bluetooth Special Interest Group (SIG), е драстично по-малко енергозависима. Твърди се, че типичните подчинени възли, работещи с Bluetooth LE, могат да работят в продължение на месеци или дори години на батерия [1, 4].

2.4.1 Предимства

Редица безжични технологии с ниска мощност могат да се използват за RF (Radio Frequency) връзка с малък обseg. Комбинацията от атрибути прави Bluetooth LE привлекателна технология за дизайнери на IoT устройства и на други безжични продукти с малък обseg. Атрибутите, които предоставя технологията са:

- Простота - радиото се базира на познатото класическо Bluetooth радио и неговият опростен режим на работа може да се управлява от малък стек протоколи;
- Ниска мощност - радиото е проектирано да прекарва почти цялото време в power-down режим, като извлича незначителен ток - дори в активен режим на предаване, той извлича нисък пиков ток, който е подходящ за работа на малка батерия;
- Здрава радиостанция - дизайнерите могат да насочват обхват на видимост от 100 метра. Схемата за прескачане на честота (frequency-hopping) прави Bluetooth LE радиото изключително устойчиво на смущения в неговия 2.4GHz нелицензиран обхват;

- Работа в реално време - може да се осъществи връзка между възел и главно устройство, прехвърляне на данни и прекъсване на връзката само за 3 ms;
- Съвместимост - Bluetooth е най-известната технология за частни мрежи в света и е подкрепена от всички основни производители на потребителски устройства.

Bluetooth SIG е проектирал Bluetooth LE съвместимост с Classic Bluetooth, така че съвместимостта с BLE може да бъде добавена евтино към устройства, които поддържат Classic Bluetooth. Например Bluetooth LE използва същата честотна лента от 2,4 GHz като Classic Bluetooth и същата антена. Така че производителите на класически Bluetooth чипове могат да добавят поддръжка за Bluetooth LE към своите продукти на много ниски цени.

Другите два познати индустриални стандарта за RF връзка с къс хоп са ZigBee и Wi-Fi, и двата имат силна привлекателност към различни приложения. ZigBee осигурява гъвкава мрежова архитектура, идеална за сложни и многовъзлови мрежи, а Wi-Fi поддържа много висока скорост на предаване на данни. Но и двете не могат да се сравнят с ниската цена, простотата и свръхниската консумация на енергия на Bluetooth LE в приложения, които предават малки количества данни.

2.4.2 Недостатъци

Освен множеството предимства, които предоставя BLE технологията, тя притежава и няколко недостатъка.

Един от недостатъците е, че Bluetooth LE не предлага възможност за класически Bluetooth в по-енергийно ефективна форма. Това не позволява на инженерите-дизайнери да постигнат функционалността на Classic Bluetooth без силно източване на батерията.

Ако дизайнът изисква класическа функционалност на Bluetooth, като поточно предаване на аудио или други данни, поддържащи скорост на предаване на данни от 1Mbps или по-висока, ще трябва да се осигури същия бюджет с висока мощност.

Bluetooth LE е насочен към съвсем различен набор от приложения. Технологията поддържа изключително бързо свързване, предаване на кратки последователности от малки пакети, последвано от бързо изключване. Тази възможност за осигуряване на „импулсно“ предаване на данни е подходяща за предаване на фрагменти от данни, като например промяна на температурата на околната среда или отчитане на движение в помещение.

Дизайнът на работата на Bluetooth LE, разбира се, не е случаен: комбинацията от способност за импулсно предаване и прескачането през честотните канали, което позволява бързи предавания без смущения, означава, че радиото може да прекарва много дълги периоди в режим на дълбок сън.

Това също така означава, че Bluetooth LE трябва да се използва за предаване на данни за състоянията, а не за поточно съдържание. Bluetooth LE устройствата могат да определят условна максимална скорост на предаване на данни в битове в секунда. Добро приложение за Bluetooth LE е това, при което Bluetooth радиото ще бъде изключено почти през цялото време.

Глава 3. Съществуващи решения

Реализациите на приложения за симулиране на работата на устройства с BLE технологията са ограничени, а повечето са разработени като емулятори.

Емуляторите и симулаторите позволяват провеждането на експерименти в софтуерно дефинирани среди. По този начин те позволяват провеждането на експерименти по-бързо, отколкото ако трябва да се настрои истинско хардуерно устройство.

Въпреки че симулаторите и емуляторите служат за сходни цели, те не работят по идентичен начин.

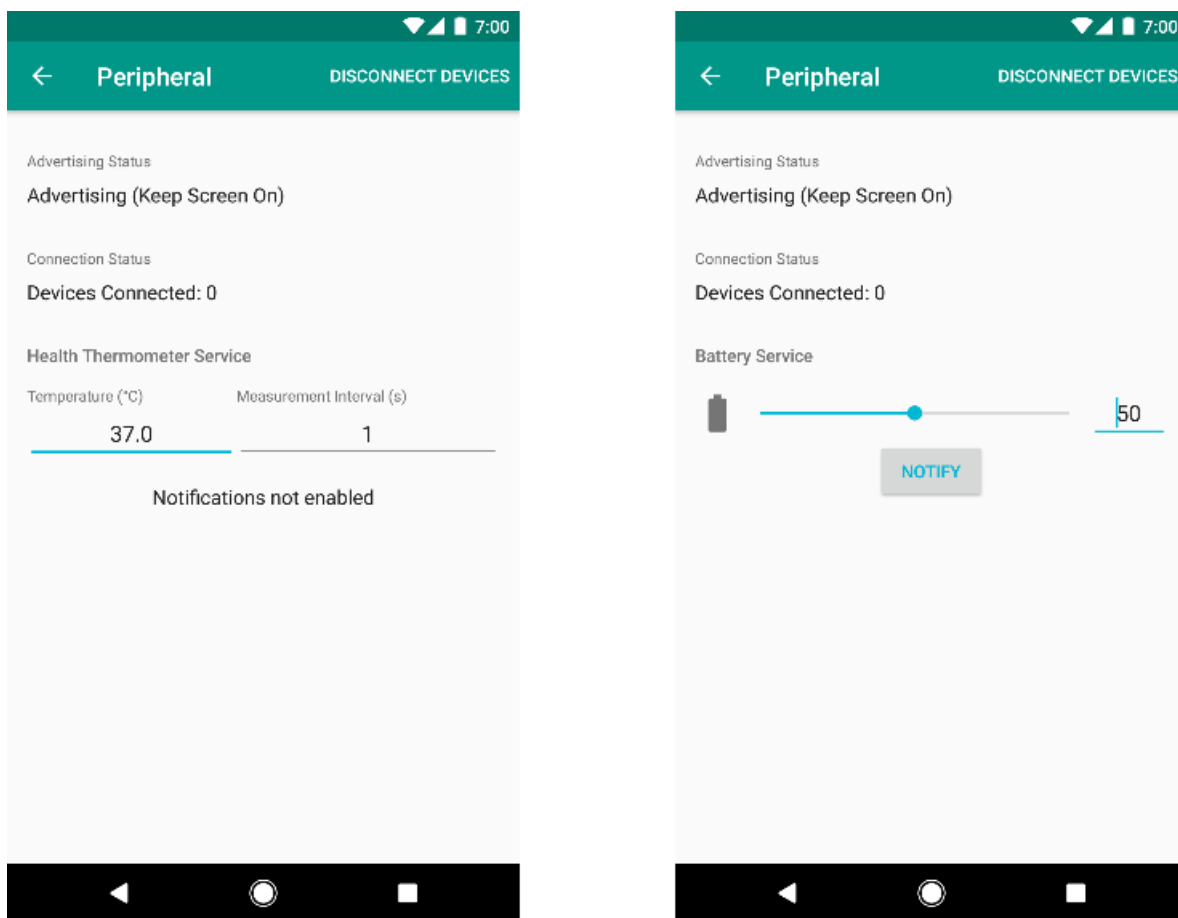
Симулаторът е проектиран да създаде среда, която съдържа всички софтуерни променливи и конфигурации, които ще съществуват в действителната производствена среда на приложението. Въпреки това, симулаторите не се опитват да емулират действителния хардуер, който ще хоства приложението в употреба. Тъй като симулаторите създават само софтуерна среда, те могат да бъдат внедрени с помощта на езици за програмиране на високо ниво. Емуляторът се опитва да имитира всички хардуерни характеристики на производствената среда, както и софтуерните функции.

3.1 BLE Peripheral Simulator

„BLE Peripheral Simulator“ е приложение за Android (Фиг. 21) услуги [5], което позволява на разработчиците да изпробват нови функции на Web Bluetooth, без да е необходимо BLE периферно устройство. За стартирането на приложението може да се използва програмния код и да се компилира или да се инсталира от Google Play Store.

Може да се използва приложението, за симулиране на BLE периферия с една от трите услуги:

- Обслужване на батерията;
- Услуга за сърдечен ритъм;
- Услуга за здравен термометър;



Фиг. 21. Потребителски интерфейс на „BLE Peripheral Simulator“

Разработчикът може да използва новите функции на Web Bluetooth, за да се свърже с приложението за четене и запис на характеристики, абониране за известия за промяна на характеристиките и четене и записване на дескриптори. Той също може да задава стойностите на характеристиките, да изпраща известия и да прекъсва връзката.

Недостатъците са, че трябва да се използва реално физическо устройство, което да притежава Bluetooth модул, и да поддържа BLE. Също има ограничени възможности, до това да може да изпраща само 3 вида.

3.2 Bluetooth Low Energy Communication интерфейса на MATLAB

BLE интерфейса в MATLAB [6] може да се използва за предаване и получаване на ASCII и двоични данни. За да е възможно това, той използва Bluetooth сериен порт. Може да се идентифицира всяко Bluetooth устройство и да се установи двупосочна връзка с това устройство от MATLAB. За да се

комуникира с BLE устройство, компютърът, на който работи MATLAB, трябва да има вграден или външен Bluetooth адаптер.

След като се свърже устройството, MATLAB се използва за четене или записване на данни към него. Може да се работи както със стандартни, така и с персонализирани услуги, характеристики и дескриптори.

Функционалностите му са следните:

- Сканиране на близки BLE периферни устройства;
- Свързване с BLE периферно устройство;
- Достъп до характеристика на периферно устройство с BLE технология;
- Достъп до дескриптор на периферно устройство с BLE технология;
- Четене на характеристики или дескрипторни данни на периферно устройство с BLE технология;
- Запис на данни в характеристика или дескриптор на периферно устройство с BLE технология;
- Абониране за характерно известие или индикация;
- Отписване от характерно известие и индикация.

Недостатъците са, че е необходимо използването на реално физическо устройство с Bluetooth модул, който да поддържа BLE, и да бъде инсталиран MATLAB на съответното устройство.

Глава 4. Избор на среда за създаване на симулатор на BLE технология

4.1 Спецификация на изискванията към приложението и цели

Настоящата дипломна работа има за цел да създаде сумулационен продукт за изграждане на BLE мрежа, който не изисква физически устройства и може да се използва за изследване на основните функционалности на технологията.

Основните дейности са:

- Добавяне и премахване на периферни устройства;
- Промяна на състоянията на периферните устройства;
- Свързване на периферни устройства с главното устройство;
- Изпращане на даннови пакети;
- Следене на изпратените пакети и данните за тях;
- Визуализиране на структурата на пакетите;
- Статистическо представяне на информация за пакетите и устройства.

Целите на симулационния продукт трябва да са:

- Да е лесен и интуитивен за ползване;
- Резултатите от работата му да се доближават до тези на физическите устройства;
- Да предоставя данните в коректен вид;
- Да може да работи в реално време;
- Да има възможност за въвеждане на различни характеристики;
- Да работи на различни операционни системи и хардуерни конфигурации;
- Изискванията му за ресурси да са ниски.

4.2 Използвани технологии

4.2.1 Десктоп приложение

Десктоп приложението е софтуерна програма, която може да се стартира на самостоятелен компютър за изпълнение на конкретна задача от краен потребител.

Едни от основните характеристики на настолните приложения са:

- Разработени са за работа на определена операционна система като Windows, Mac или Linux;
- Актуализациите на настолните приложения трябва да бъдат инсталирани от крайните потребители. Актуализациите могат да бъдат публикувани чрез Интернет, но инсталацията обикновено е ръчен процес, извършен от крайния потребител;
- Приложенията за настолни компютри са проектирани да работят в изолирана среда, поради което имат по-малко проблеми със сигурността;
- Възможността за работа без интернет връзка е друга често срещана характеристика на настолните приложения.

4.2.2 Java

Java е обектно-ориентиран език за програмиране и е проектиран да има възможно най-малко зависимости от изпълнението. Той е език за програмиране с общо предназначение, създаден с цел разработчиците да пишат код веднъж, който да се изпълнява навсякъде където Java код може да работи. Java приложенията се компилират в байтов код, който може да се изпълнява на всяка Java виртуална машина [7].

Основни характеристики на Java:

- Независим от платформата: компилаторът преобразува изходния код в байт код и след това JVM (Java Virtual Machine) изпълнява байт кода, генериран от компилатора. Този байт код може да работи на всяка платформа, било то Windows, Linux, macOS, което означава, че ако компилираме програма на Windows, тогава можем да я стартираме на Linux и обратно. Всяка операционна система има различен JVM, но

изходът, произведен от всички ОС, е един и същ след изпълнението на байт кода. Ето защо Java се нарича независим от платформата език;

- **Обектно-ориентиран език за програмиране:** организирането на програмата по отношение на събирането на обекти е начин на обектно-ориентирано програмиране, всеки от които представлява екземпляр на класа.

- **Простота:** Java е един от простите езици, тъй като няма сложни функции като указатели, множество наследявания, изрично разпределение на паметта;

- **Стабилност:** езикът Java е стабилен, което означава надежден. Той е разработен по такъв начин, че полага много усилия за проверка на грешки възможно най-рано, затова компилаторът на java е в състояние да открие дори тези грешки, които не са лесни за откриване от друг език за програмиране. Основните характеристики на java, които я правят стабилна, са „събирача на боклук“ (garbage collector), обработка на изключения и разпределение на паметта;

- **Сигурност:** в Java няма указатели и затова не може да се получи достъп до клетки извън масиви, т.е. той показва „ArrayIndexOutOfBoundsException“, при опит да се направи. Ето защо няколко недостатъка в сигурността като повреда при стека или препълване на буфера е невъзможно да се използват в Java;

- **Многопоточност:** Java поддържа многопоточност. Това е функция, която позволява едновременно изпълнение на две или повече части на програма за максимално използване на процесора;

- **Преносимост:** всеки Java код може да се изпълнява на друга машина, независимо от операционната ѝ система.

4.2.3 Apache Maven

Maven е инструмент за управление на проекти, който се основава на Project Object Model (POM). Използва се за изграждане на проекти, зависимости и документация [8].

Накратко maven е инструмент, който може да се използва за изграждане и управление на който и да е Java базиран проект. Той улеснява ежедневната работа на разработчиците на Java и като цяло помага за разбирането на всеки Java базиран проект.

Основни концепции на Maven:

- **РОМ файлове:** файловете на обектния модел на проекта (РОМ) са XML файлове, които съдържат информация, свързана с проекта и информация за конфигурацията, като зависимости, директория на източника, плъгин, цели и т.н., използвани от Maven за изграждане на проекта. Когато трябва да се изпълни maven команда, му се подава РОМ файла за изпълнение на командите. Maven чете pom.xml файла, за да изпълни своята конфигурация и операции;
- **Зависимости (dependencies) и хранилища (repositories):** зависимостите са външни библиотеки на Java, необходими за проекта, а хранилищата са директории на пакетирани JAR файлове. Локалното хранилище е просто директория на твърдия диск на машина, на която се изпълнява проекта. Ако зависимостите не бъдат намерени в локалното хранилище на Maven, той ги изтегля от централно хранилище и ги поставя в локалното;
- **Жизнен цикъл на изграждане, фази и цели:** жизнения цикъл на изграждането се състои от последователност от фази на изграждане, а всяка фаза на изграждане се състои от последователност от цели;
- **Профили на изграждане:** те са набор от конфигурационни стойности, които позволяват да се изгради проекта, използвайки различни конфигурации. Например може да се наложи да се създаде проект за локален компютър, за разработка и тестване. За да се разрешат различни компилации, може да се добавят различни профили за компилация към РОМ файловете, като се използват елементите на профилите и се задействат по различни начини;
- **Приставки за изграждане (Build Plugins):** приставките за изграждане се използват за изпълнение на конкретна цел. Може да се добави приставка към РОМ файла. Maven има някои стандартни приставки, които може да се използват, а също така може и да се внедрят собствени в Java.

4.2.4 JavaFX & Scene Builder

JavaFX е библиотека на Java [9], използвана за разработване на настолни приложения, както и интернет приложения. Приложенията, използващи JavaFX, могат да работят на множество платформи, включително уеб, мобилни и настолни компютри.

JavaFX е предназначен да замени Swing в приложенията на Java като GUI рамка. Той обаче предоставя повече функционалности, отколкото Swing. Подобно на Swing, JavaFX също предоставя свои собствени компоненти и не зависи от операционната система. Той е лек и хардуерно ускорен и поддържа различни операционни системи, включително Windows, Linux и Mac OS.

4.2.4.1 Характеристики на JavaFX

- Java библиотека - това е библиотека на Java, която се състои от много класове и интерфейси, написани на Java;
- FXML - базиран на XML декларативен език за маркиране. Кодирането може да се извърши във FXML, за да предостави подобрения GUI на потребителя;
- Scene Builder - генерира FXML код, който може да бъде пренесена в IDE;
- Уеб изглед - уеб страниците могат да бъдат вградени в JavaFX приложения. Web изгледа използва технологията WebKitHTML за вграждане на уеб страници;
- Вградените контроли за потребителски интерфейс - JavaFX съдържа вградени компоненти, които не зависят от операционната система. Компонентите на потребителския интерфейс са достатъчни, за да разработят пълнофункционално приложение;
- CSS стилизиране - JavaFX код може да бъде вграден в CSS за подобряване на стила на приложението. Може да се подобри изгледа на приложението с прости познания за CSS;
- Интегрирана графична библиотека - предлагат се интегриран набор от класове за работа с 2D и 3D графика;
- Графичен конвейер – графиките на JavaFX се основават на графично визуален конвейер. Той предлага плавна графика, която е хардуерно ускорена;
- Самостоятелен модел за внедряване на приложения - самосъдържащите се пакети за приложения имат всички ресурси на приложението и частно копие на Java и JavaFX Runtime.

4.2.4.2 Scene Builder

JavaFX Scene Builder (Scene Builder) позволява бързо да се проектират потребителски интерфейси на JavaFX приложение, поради факта, че има drag-and-drop функционалност, чрез която може да се наместват ръчно UI елементи върху областта на изглед. FXML кодът за оформлението на потребителския интерфейс, който се създава в инструмента, се генерира автоматично във фонов режим.

Scene Builder може да се използва като самостоятелен инструмент за проектиране, но може да се използва и заедно с Java IDE, така че да може да се използва IDE за писане, изграждане и стартиране на изходния код на контролера, който се използва с потребителския интерфейс на приложението. Въпреки че Scene Builder е по-тясно интегриран с IDE на NetBeans, той е интегриран и с други Java IDE.

4.2.5 Launch4j

Launch4j [10] е междуплатформен инструмент за опаковане на Java приложения, разпространявани като jar файлове в изпълними (.exe) файлове на Windows. Изпълнимият файл може да бъде конфигуриран да търси определена версия на JRE или да използва пакетна и е възможно да се зададат опции за изпълнение, като първоначален / максимален размер на heap паметта. Опаковането също така осигурява по-добро потребителско изживяване чрез икона на приложение, естествен начален екран преди JRE и препращане към страница за изтегляне на Java, в случай че не може да бъде намерен подходящият JRE на компютъра на който се стартира изпълнимия файл.

Някои от неговите свойства са:

- Launch4j обвива jar файловете в изпълними файлове на Windows и позволява да се стартират като обикновена програма на Windows. Възможно е да се опаковат приложения на Windows, Linux и Mac OS X;
- Работи с включен JRE или търси най-новите Sun или IBM JRE / JDK в даден обхват на версиите и тип на процесора (64-битов или 32-битов);
- Отваря страницата за изтегляне на Java, ако не може да бъде намерена подходяща версия на Java или уебсайт за поддръжка в случай на грешка;

- Поддържа GUI и конзолни приложения;
- Възможност за рестартиране на приложението въз основа на изходния код;
- Интеграция на изграждане чрез Maven;
- Безплатен е и може да се използва за комерсиални цели.

Глава 5. Структура на създавания симулатор

5.1 Архитектура на директориите

Проекта се разделя на 8 пакета, като шаблона за проектиране е MVC (Model – View - Controller). Концепцията му е да се раздели приложението на три основни логически части – модел, изглед и контролер. Моделът съдържа информация за данните, изгледът отговаря за презентацията на данните от модела, а контролерът обработва данните от модела и ги подава за визуализиране от изгледа. Всеки един от тези модули е създаден, за да управлява специфична част от софтуера. Пакетите са (Фиг. 22):

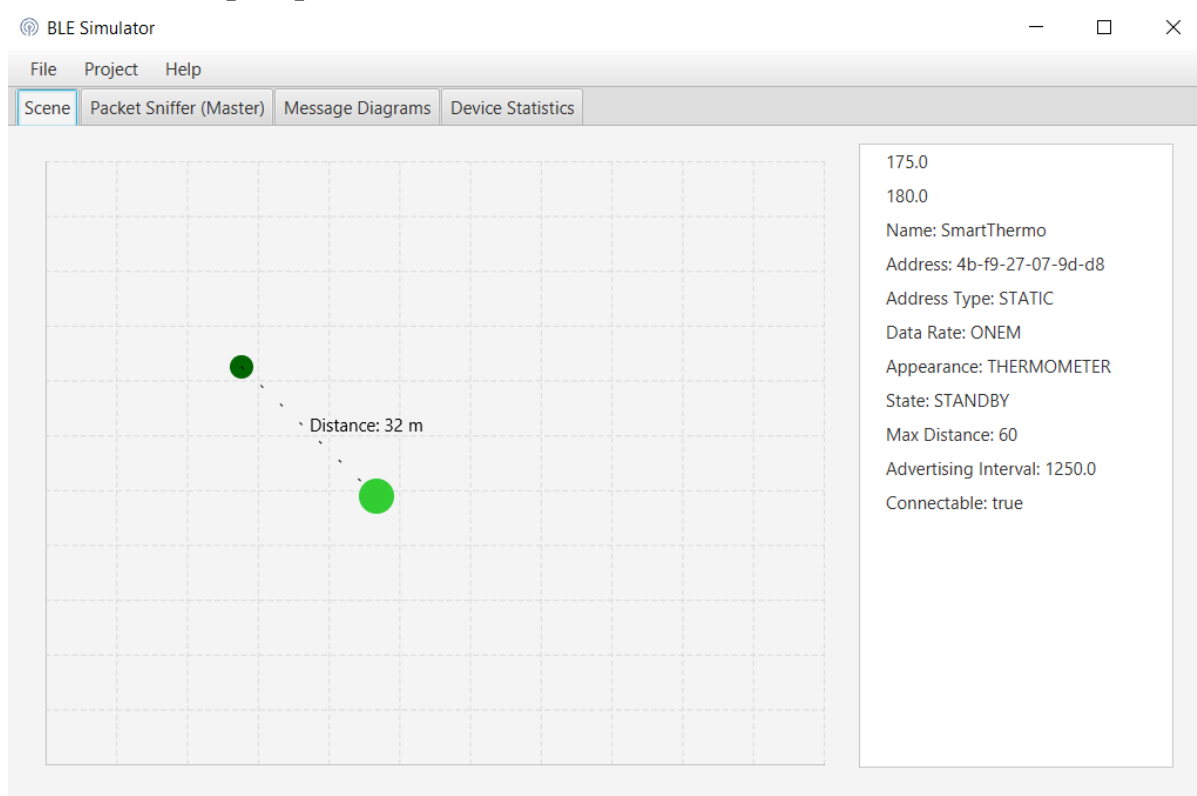
- “app” – съдържа стартовата точка на програмата “SuperMain”, която служи за извикване на стартиращата функция от класа “App”;
- „controller“ – съдържа контролерите на различните fxml файлове;
- “images” – съдържа изображения необходими за програмата;
- “model” – съдържа моделът на обекта за устройствата;
- “packets” – съдържа класът “Packet”, който описва пакетните обекти;
- “utilities” – съдържа помощни класове, които извършват различни функционалности;
- “validations” – класът “InputValidator” съдържа статични функции, които служат за валидации при въвеждане на полета в програмата;
- “view” – съдържа fxml файловете, които са базирани на xml и служат за визуализиране на дизайна. Също така в този пакет се намират и езиковите файлове за български и английски.

- ▼ 📁 src/main/java
 - ▼ 📁 app
 - > 📄 App.java
 - > 📄 SuperMain.java
 - ▼ 📁 controller
 - > 📄 AddDeviceController.java
 - > 📄 AppController.java
 - > 📄 ConnectionController.java
 - > 📄 PacketController.java
 - > 📄 SendPacketController.java
 - > 📁 images
 - ▼ 📁 model
 - > 📄 AddressType.java
 - > 📄 DataRate.java
 - > 📄 Device.java
 - > 📄 DeviceAddress.java
 - > 📄 DeviceCircle.java
 - ▼ 📁 packets
 - > 📄 Packet.java
 - > 📄 PacketType.java
 - > 📄 Services.java
 - ▼ 📁 utilities
 - > 📄 ConnectionUtil.java
 - > 📄 DevicePacketFactory.java
 - > 📄 DeviceStatisticsUtil.java
 - > 📄 MasterPacketFactory.java
 - > 📄 MessageSequenceFactory.java
 - > 📄 ScenePaneUtil.java
 - > 📄 Singleton.java
 - ▼ 📁 validations
 - > 📄 InputValidator.java
 - > 📁 view

Фиг. 22. Архитектура на пакетите на проекта

5.2 Потребителски интерфейс

5.2.1 Главен прозорец



Фиг. 23. Начален прозорец на симулатора

Главния прозорец на програмата се визуализира при стартирането ѝ. На нея са разположени меню лентата и лента за табове, като всеки таб изобразява различна функционалност на програмата.

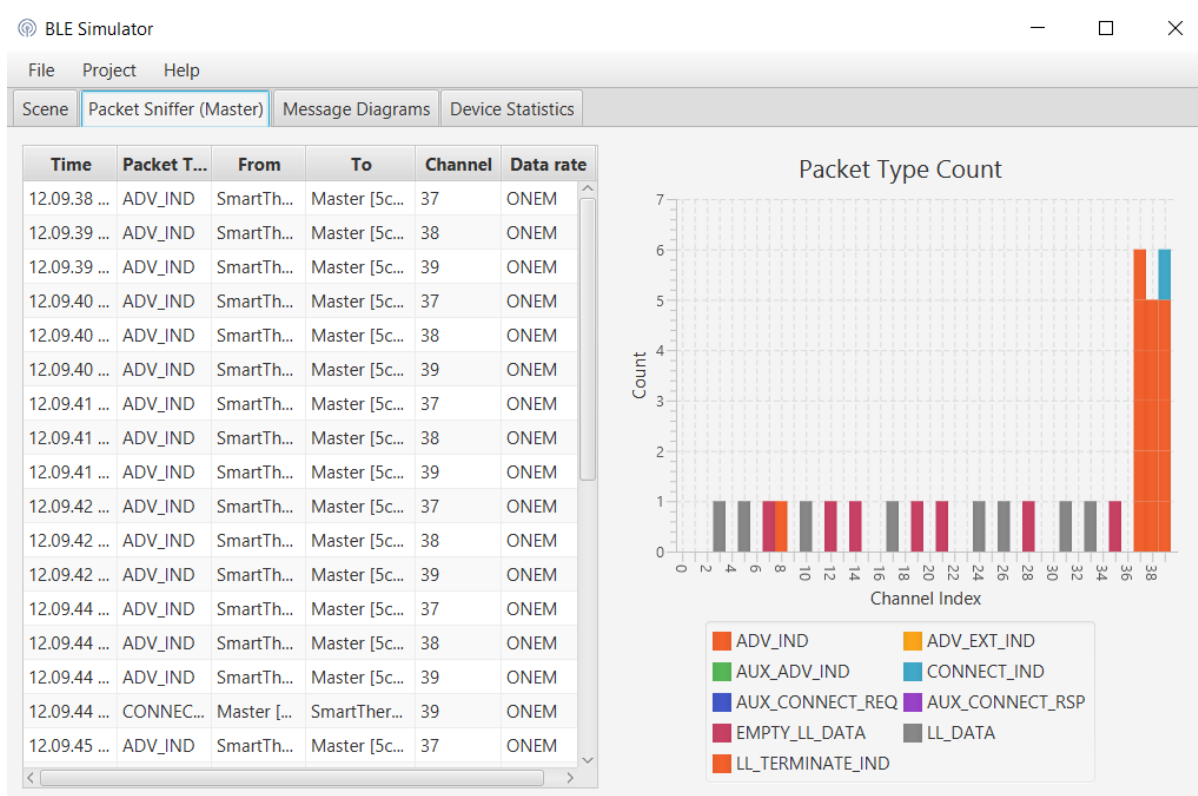
5.2.1.1 Табът Scene

На него се разполагат елементите за визуализиране на топологията и поле за информация на избраното устройство.

За визуализирането на топологията се използва Pane елемент, на който е добавена функционалността да могат да се местят графичните елементи на устройствата – кръговете. При разместване ако разстоянието между устройствата е в посочените граници, то ще се изобрази пунктирна линия, която индикира, че двете устройства могат да комуникират едно с друго. В допълнение ще се изобрази с текст самото разстояние.

При избор на графичен елемент на устройство, в полето отляво ще се изобрази информация за него, като име, адрес, максимално разстояние и д.р.

5.2.1.2 Табът Packet Sniffer (Master)

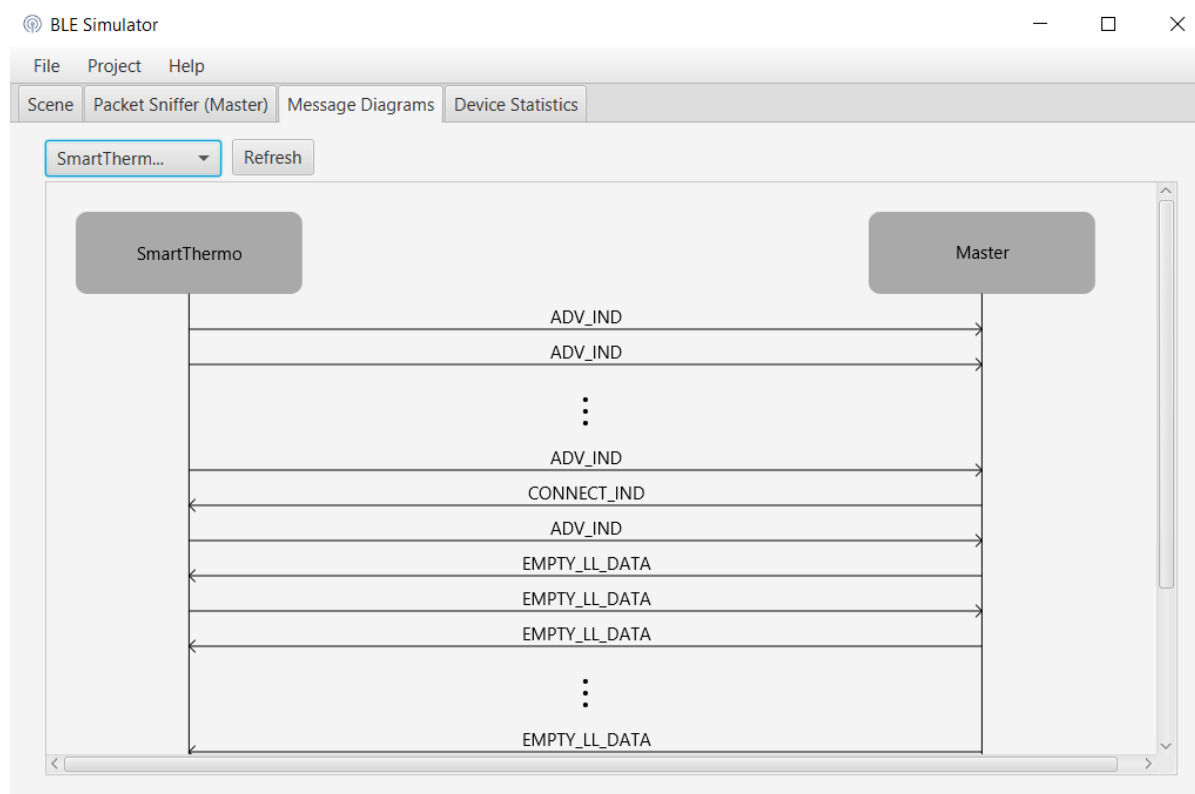


Фиг. 24. Изглед на табът за показване на пакетите при главното устройство

На този таб се показва информация за това какви пакети постъпват на главното устройство и какво той изпраща и диаграма за визуализиране на бройката на различните видове пакети, които са показани в таблицата.

В таблицата за пакетите са показани данни за часът и датата на изпращане на пакета, типът на пакета, от кого и до кого е, на кой канал е изпратен и какъв е неговия data rate.

5.2.1.3 Табът Message Diagrams

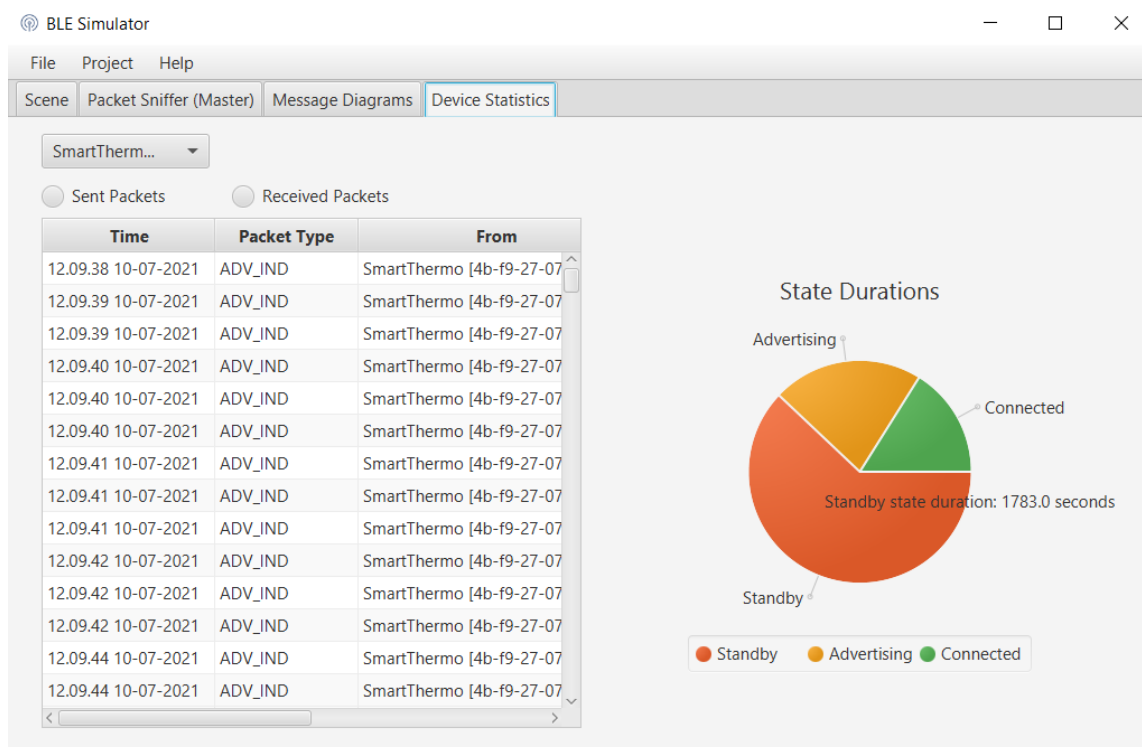


Фиг. 25. Изглед на табът за показване на последователността от съобщения между избрано устройство и главното устройство

Служи за представяне на диаграма за последователността на съобщенията, като се показва какъв тип пакет се изпраща от кое за кое устройство, в удобен за четене вид.

5.2.1.4 Табът Device Statistics

Показва информация за периферните устройства, като получени и изпратени пакети и също така визуализира диаграма за време на работа във състоянията му на рекламиране, в режим на готовност и на връзка.



Фиг. 26. Изглед на табът за показване на детайли за крайните устройства

5.2.1.5 Добавяне на ново устройство

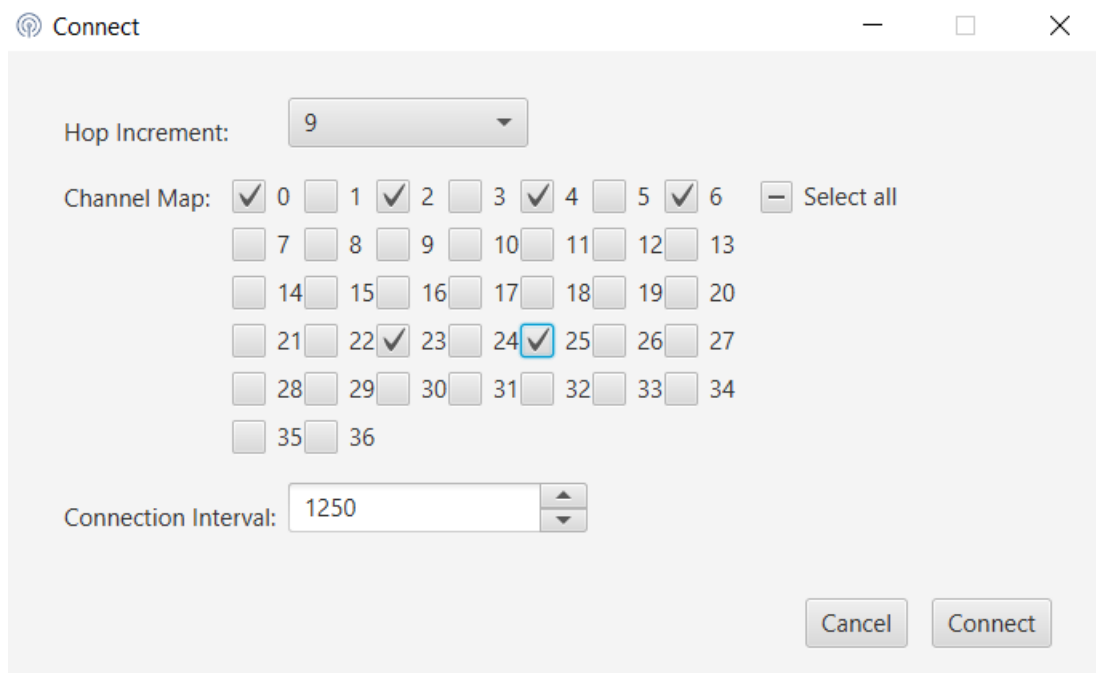
При добавянето на устройство се въвежда информация за името му, типът на адреса му, дали може да бъде свързано или не, какъв е неговия data rate, неговия изглед, интервал на рекламиране и максимално разстояние, при което ще има комуникация между него и главното устройство.

The 'Add Device' dialog box contains the following fields and options:

- Device name:** SmartThermo
- Address type:**
 - ☐ Public
 - ☐ Static
 - ☐ Resolvable
 - ☒ Non Resolvable d7-93-56-8c-d0-d4
- Connectivity:**
 - ☒ Connectable
 - ☐ Non-Connectable
- Data rate:** TWOM
- Appearance:** THERMOME...
- Advertising interval:** 1250
- Max Distance:** 60 m
- Buttons:** Cancel, Add Device

Фиг. 27. Изглед на прозорец за добавяне на крайни устройства

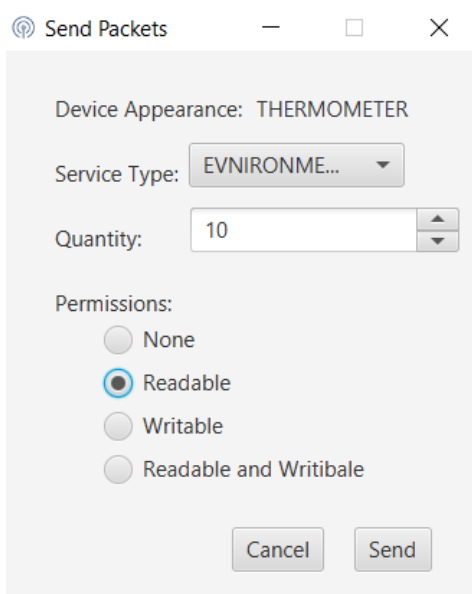
5.2.1.6 Свързване на периферно устройство с главното



Фиг. 28. Изглед на прозорец за свързване на периферно и главно устройство

За свързването на устройствата се изисква информация за това какъв е hop инкремента, мапинга на каналите и интервала на връзка, който указва през колко време се повтаря събитието на връзка.

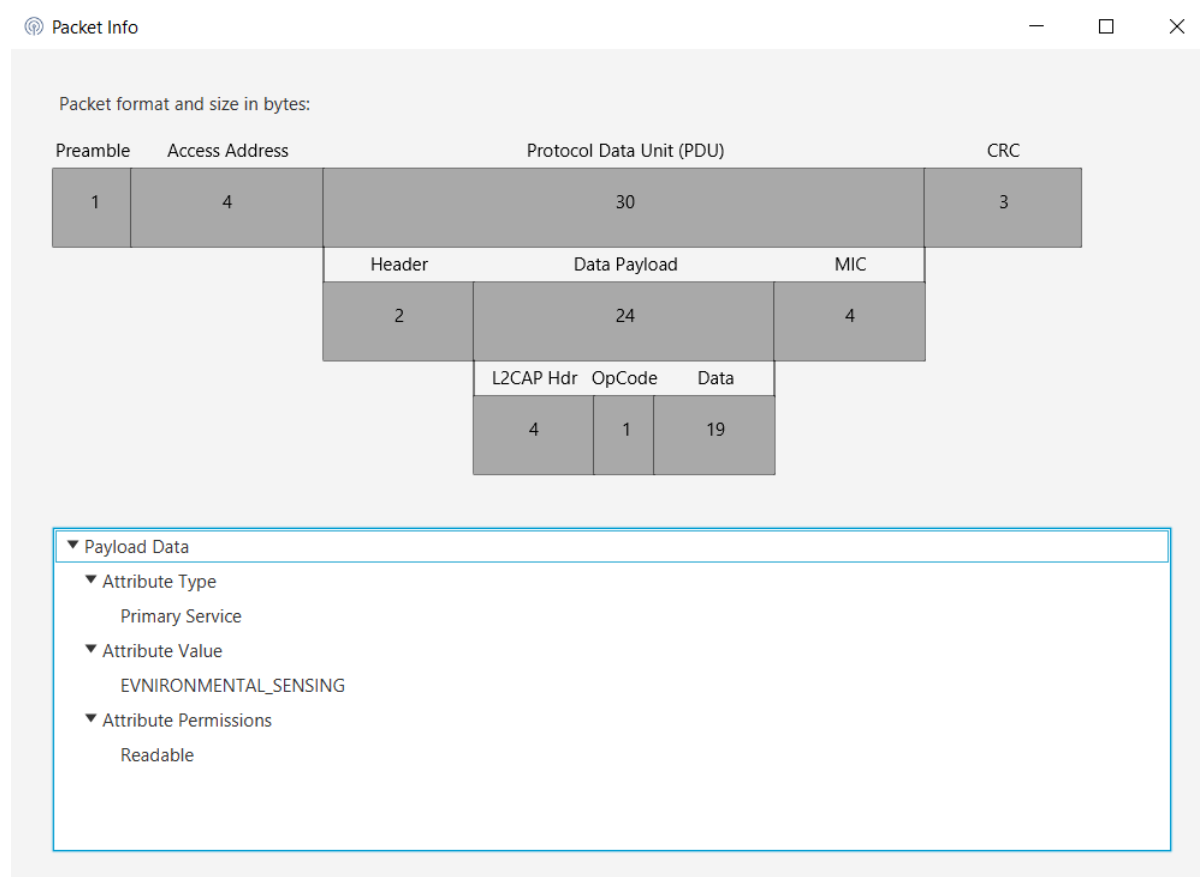
5.2.1.7 Изпращане на даннови пакети от подчиненото към главното устройство



Фиг. 29. Изглед за изпращане на даннови пакети

При изпращането на пакети се изобразява какъв е изгледът на устройството и от това зависи какъв вид пакет на услугите ще може да бъде изпратен от тези въведени в програмата. Друга информация, която се въвежда е бройката на тези пакети и какви са разрешенията за работа с данните от пакетите, които ще се изпращат.

5.2.1.8 Информация за пакетите



Фиг. 30. Изглед на структура на пакета

При изобразяването на допълнителна информация на пакетите, се генерира формата на пакета и размера на отделните му елементи. Формата за всеки тип пакети е различен, според дефиницията му в Bluetooth класификацията. В полето под Фигурата се изобразява допълнителна информация за данните вътре в пакета.

5.3 Работа на приложението

При стартиране на програмата се зарежда главния прозорец, в който са разположени повечето функционалности. Този прозорец се контролира от класа

„AppController“, като той отговаря за функционалността на меню лентата, добавянето на главното устройство и периферните устройства, таблицата за получените и изпратените от него пакети и диаграмата за съответния Protocol Data Unit тип на тези пакети.

Функции за добавяне на главно и периферно устройство:

```
private void addMaster() {
    ScenePaneUtil.makeDragable(master, scenePane);
    ScenePaneUtil.onClickListener(master, scenePane, detailsTreeView);

    this.scenePane.getChildren().add(master.getDeviceCircle().getCircle());
    master.getDeviceCircle().getCircle().setLayoutX(300);
    master.getDeviceCircle().getCircle().setLayoutY(300);

    MasterPacketFactory.listenForReceivedPacketsOnMaster(master);
}

public void addDevice(Device device) {
    Singleton.getInstance().executor.execute(device.getPacketFactory());

    devices.add(device);
    ScenePaneUtil.makeDragable(device, scenePane);
    ScenePaneUtil.onClickListener(device, scenePane, detailsTreeView);

    this.scenePane.getChildren().add(device.getDeviceCircle().getCircle());
    this.scenePane.getChildren().add(device.getDeviceCircle().getLine());

    this.scenePane.getChildren().add(device.getDeviceCircle().getText());
    device.getDeviceCircle().getLine().setVisible(false);

    device.getDeviceCircle().getCircle().setLayoutX(100);
    device.getDeviceCircle().getCircle().setLayoutY(100);
}
```

Функция за зареждане на таблицата с пакети на главното устройство:

```
public void loadTableView() {
    this.packetTableView.setRowFactory(tv -> {

        TableRow<Packet> row = new TableRow<>();
        row.setOnMouseClicked(event -> {
            if (event.getClickCount() == 2 && (!row.isEmpty())) {
                Parent root;
                try {
                    PacketController.setPacket(row.getItem());
                    root =
FXXMLLoader.load(getClass().getResource("/view/PackagePane.fxml"));
                    Stage stage = new Stage();
                    stage.setTitle("Packet Info");
                    stage.setScene(new Scene(root, 1000, 700));
                    stage.getIcons().add(new
Image(getClass().getResourceAsStream("/images/logo.png")));
                    stage.setResizable(false);
                    stage.show();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        });
    });
    return row;
}
```

```

});

    TableColumn<Packet, String> timeColumn = new TableColumn<>("%timeLabel");
    timeColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("timeLabel"));
    timeColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("time"));
    timeColumn.setSortType(TableColumn.SortType.DESCENDING);

    ...

    this.packetTableView.getColumns().setAll(timeColumn, typeColumn, fromColumn,
toColumn, channelColumn,
                                dataRateColumn);

    this.packetTableView.setItems(master.getAllPackets());
}

```

Функция за зареждане на диаграмата за броене на типовете пакети:

```

public void loadChannelBarChart() {
    XYChart.Series<String, Integer> advInd = new XYChart.Series<>();
    advInd.setName("ADV_IND");

    ...

    master.getAllPackets().addListener(new ListChangeListener<Packet>() {

        @Override
        public void onChanged(Change<? extends Packet> c) {
            Packet lastPacket = c.getList().get(c.getList().size() - 1);

            Platform.runLater(() -> {
                switch (lastPacket.getPacketType()) {
                    case ADV_IND:
                        advInd.getData().add(new
XYChart.Data<>(lastPacket.getChannel(), 1));
                        break;

                        ...

                    default:
                        break;
                }
            });
        }
    });

    for (int i = 0; i < 40; i++) {
        advExtInd.getData().add(new XYChart.Data<String,
Integer>(Integer.toString(i), 0));
    }

    this.channelStackedBarChart.setCategoryGap(0);
    this.channelStackedBarChart.getData().setAll(advInd, advExtInd, auxAdvInd,
connectInd, auxConnectReq, auxConnectRsp, emptyLLdata, llData, terminateIND);
}

```

Добавянето на периферни устройства е възможно, чрез избор от меню лентата „Project -> Add Device“. Появява се диалогов прозорец, където може да се зададат параметрите на това устройство, като име, тип на адрес, дали може да се осъществи връзка с него или не, неговия Data rate, изглед (служи за

асоцииране на устройството с обща категория), интервал на рекламиране и максимално разстояние, при което може да има обмен на пакети между него и главното устройство. Тази функционалност се контролира от класа „AddDeviceController“, в който се обработва зададената в прозореца информация. Работата на добавеното устройство се контролира от „DevicePacketFactory“ класа. Той отговаря за състоянието на устройството и функционалността на всяко състояние. Състоянията са рекламиране, в готовност и свързан.

Функция за създаване на устройство в „AddDeviceController“:

```
public void addDevice() {
    if (validateInput()) {
        errorLabel.setVisible(false);
        if (selectedAddressType != null) {
            if (selectedAddressType.equals(AddressType.PUBLIC)) {

                deviceAddress.setAddress(publicAddressTextField.getText());
            }

            Circle circle = new Circle();
            circle.setRadius(10);
            circle.setId(deviceNameTextField.getText());
            circle.setFill(Color.valueOf(AppController.getRandomColorName()));

            DeviceCircle deviceCircle = new DeviceCircle(circle, new
Line());

            newDevice = new Device(deviceNameTextField.getText(),
deviceAddress, "STANDBY", dataRateComboBox.getSelectionModel().getSelectedItem(),
appearanceComboBox.getSelectionModel().getSelectedItem(), deviceCircle);

            newDevice.getPacketFactory().setAdvertisingInterval(advertisingIntervalSpinner.getV
alue().toString());

            newDevice.getPacketFactory().setMaxDistance(maxDistanceTextField.getText());
            newDevice.getPacketFactory().setConnectable(((RadioButton)
connectableGroup.getSelectedToggle()).getText().equals("Connectable") ? true:
false);

            cancel();
        }
    } else {
        System.out.println("Input fields!");
        errorLabel.setVisible(true);
    }
}
```

В „DevicePacketFactory“ функцията, която ще се изпълнява от нишка е:

```
@Override
public void run() {
    synchronized (lock) {
        try {
            switch (device.getState()) {
                case ADVERTISING:
                    advertisementEvent();
                    this.run();
                    break;
            }
        }
    }
}
```

```

        case CONNECTION:
            connectionEvent();
            this.run();
            break;
        case STANDBY:
            lock.wait();
            this.run();
            break;
        case REMOVAL:
            break;

        default:
            break;
    }
} catch (InterruptedException e) {
    e.printStackTrace();
}
}

```

Когато методът „run()“ се извика, кодът, посочен в него, се изпълнява и също може да се извиква многократно. С помощта на „ThreadPoolExecutor“ ще се извиква непрекъснато този метод, върху която и да е свободна нишка, но ако в дадения момент няма свободна – задачата се поставя в опашка за чакане.

Събитието за рекламиране включва в себе си функционалността за рекламиране, която отговаря за формирането на рекламен пакет (advertisement packet) според data rate на устройството, понеже формата и бройката на пакети се различава в зависимост от това, и след това изпраща пакета към главното устройство, ако то е в обсега му. След това се проверява за получени пакети за инициране на връзка от главното устройство, и ако има такива да се направят съответните действия. Следващото повторение на събитието за рекламиране се случва на следващия канал за рекламиране от дефинираните в Bluetooth стандарта.

Функцията има вида:

```

private void advertisementEvent() throws InterruptedException {
    advertise();
    listenForConnectionRequest();

    Thread.sleep((long) (10 / Singleton.getInstance().speed));

    swapAdvertisingChannel();
}

```

Събитието на свързване първо проверя за получени пакети, като с цел симулиране на свързване е имплементирано да слуша за празни даннови пакети и пакет за прекратяване на връзката. След това ако потребителят е решил да изпрати пакети чрез графичния интерфейс, което може да се извърши с избор на “Send Data Packets” от контекстното меню на свързаното устройство, видимо с десен бутон на мишката. Ако няма пакети, които да чакат изпращане,

устройството изпраща празни даннови пакети, които имат за цел поддържане на връзката. Формирането на пакета, който ще се изпрати и самото му изпращане се извършва от класа „SendPacketController“. В състоянието "в готовност" устройството просто изчаква следващия цикъл на работа.

Функцията за събитие на свързване има следния вид:

```
private void connectionEvent() {
    try {

Thread.sleep(Double.valueOf(connectionController.getConnectionInterval()).longValue
());
        listenForReceivedPackets();
        dataPacketFactory();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Функциите за формиране и изпращане на даннов пакет:

```
public void send() {
    if (validateInput()) {
        errorLabel.setVisible(false);
        if (!slave.getPacketFactory().isHavingPacketsToSend()) {
            generateDataPackets();
            cancel();
        }
    } else {
        errorLabel.setVisible(true);
    }
}

private void generateDataPackets() {
    String data = "Attribute Handle:" + "\nAttribute Type:Primary Service" +
"\nAttribute Value:" + serviceComboBox.getValue() + "\nAttribute Permissions:" +
((RadioButton) permissionsGroup.getSelectedToggle()).getText();

    Packet dataPacket = new Packet(Singleton.getTime(), PacketType.LL_DATA,
slave, Singleton.getInstance().master,

    ConnectionUtil.nextChannel(slave.getPacketFactory().getConnectionController()
.getChannelMap(),
slave.getPacketFactory().getConnectionController().getHopIncrement()),
slave.getDataRate(), data);

    slave.getPacketFactory().setSendDataPackets(dataPacket,
quantitySpinner.getValue().intValue());
}
```

Работата на всяко устройство се контролира чрез нишки, за да се позволи паралелната обработка, която да доближи работата на симулатора до тази в реална среда. За работата на нишките се използва ThreadPollExecutor, който изпълнява всяка подадена задача, използвайки един от няколкото thread пулове. В приложението се използва Executors.newCachedThreadPool() метода за инстанциране на ThreadPollExecutor. Пулът се състои от автоматично променящ се брой нишки в зависимост от бройката на постъпващите задачи. Началния

брой на нишките (corePoolSize) е 0. Thread пулът помага да се спестят ресурси в многонишковото приложение и също така да се увеличи производителността.

Инициализиране на пулът с нишки:

```
public ThreadPoolExecutor executor;
...
private Singleton() {
    ...
    executor = (ThreadPoolExecutor)
Executors.newCachedThreadPool();
    ...
}

Singleton.getInstance().executor.execute(device.getPacketFactory());
```

За да се свърже главното устройство с периферно, първо се проверява дали периферното рекламира в дадения момент, ако да, то тогава връзка е възможна. Това се проверява от класа „ScenePaneUtil”. За свързване трябва от контекстното меню на периферното устройство да се избере „Connect”. В появилия се диалогов прозорец трябва да се зададе хоп инкремента, channel mapping и интервала на събитието за обмен на даннови пакети (connection interval). След като всичко е зададено, двете устройства вече биват свързани и започва изпращането на празни пакети от двете страни, за да се индикира, че има връзка и тя се поддържа. Вече има възможност да се изпратят даннови пакети от подчиненото устройство към главното.

Работа на функцията за свързване на устройства от контекстното меню:

```
ContextMenu contextMenu = new ContextMenu();

MenuItem connectDisconnectButton = new
MenuItem(device.getState().equals(Device.State.CONNECTION) ? "Disconnect" :
"Connect");

item1.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        if (item1.getText().equals("Connect")) {
            ...

            // Отваряне на прозорец за свързване

            ...

        } else {
            ...

            // Формиране на Disconnect пакет за спиране на връзката

            ...

            Packet.sendPacket(disconnectPacket);
        }
    }
});
```

```

...

if (!device.getDeviceCircle().getLine().isVisible() ||
!device.getPacketFactory().isConnectable() ||
device.getState().equals(Device.State.STANDBY)) {
    connectDisconnectButton.setDisable(true);
} else {
    connectDisconnectButton.setDisable(false);
}

contextMenu.show(device.getDeviceCircle().getCircle(), event.getScreenX(),
event.getScreenY());

```

Друга функционалност е диаграмата на съобщения, която изобразява какви видове пакети се изпращат към кое устройство, като при случай на повторение на пакетите се поставя многоточие за да не се пренасити с информация, което да я прави трудна за разбиране. Генерирането на тази диаграма става чрез класа „MessageSequenceFactory“, който отговаря за изчертаването на елементите на графиката. Самото генериране на стрелките на съобщенията и техния надпис става като се вземе списък на пакетите от главното устройство, които е обменил с избраното периферно и според това кой е изпращача на пакета се подразбира накъде да сочи стрелката. Също се прави сравнение с предишни пакети и ако се открие повторение, то се изобразява многоточие, и когато е наред пакет, който е различен от предходните повтарящи се, то той вече се изобразява.

Функция за изчертаване на диаграмата за последователност на съобщенията:

```

private static void draw() {
    if (currentSlave != null)
        makeDeviceRectangle(25, 25, currentSlave.getName());

    makeDeviceRectangle(700, 25, "Master");

    ObservableList<Packet> packets =
FXCollections.observableArrayList(Singleton.getInstance().master.getAllPackets().stream().filter(p -> p.getDeviceFrom().equals(currentSlave) ||
p.getDeviceTo().equals(currentSlave)).collect(Collectors.toList()));

    for (int i = 0; i < packets.size(); i++) {

        if ( пакета е ADV_EXT_IND или AUX_ADV_IND ) {
            if ( изпращащият и типът на пакета са същите като предишния ) {
                if (!drawedDot) {
                    drawRepeatingDots();
                    ...
                }
            } else {
                generateMessageLines(packets.get(i));
                drawedDot = false;
            }
        } else if ( пакета е EMPTY_LL_DATA ) {
            if ( има съвпадаща последователност последните 4 пакета ) {
                if (!drawedDot) {

```

```

        drawRepeatingDots();
        ...
    }
    } else {
        generateMessageLines(packets.get(i));
        drawnDot = false;
    }
} else {
    if ( пакета съвпада с предишния ) {
        if (!drawnDot) {
            drawRepeatingDots();
            ...
        }
    } else {
        generateMessageLines(packets.get(i));
        drawnDot = false;
    }
}
}
}
}

```

Има възможност и за проверяване на изпратени и получени пакети от страна на периферните устройства, както и за изобразяване на тяхното време на работа в диаграма, това става чрез класа „DeviceStatisticsUtil“.

Функция за изобразяване на диаграма за работното време в дадено състояние:

```

public static void loadPieChart() {
    PieChart.Data slice1 = new PieChart.Data("%standbyLabel", 0);
    slice1.setName(ResourceBundle.getBundle("view.lang",
    Locale.getDefault()).getString("standbyLabel"));
    PieChart.Data slice2 = new PieChart.Data("%advertisingLabel", 0);
    slice2.setName(ResourceBundle.getBundle("view.lang",
    Locale.getDefault()).getString("advertisingLabel"));
    PieChart.Data slice3 = new PieChart.Data("%connectedLabel", 0);
    slice3.setName(ResourceBundle.getBundle("view.lang",
    Locale.getDefault()).getString("connectedLabel"));

    slice1.pieValueProperty().bind(device.getStandbyTime());
    slice2.pieValueProperty().bind(device.getAdvertisingTime());
    slice3.pieValueProperty().bind(device.getConnectedeTime());

    ObservableList<PieChart.Data> data =
    FXCollections.observableArrayList(slice1, slice2, slice3);
    pieChart.setData(data);

    for (final PieChart.Data d : pieChart.getData()) {
        d.getNode().addEventHandler(MouseEvent.MOUSE_PRESSED, new
    EventHandler<MouseEvent>() {
        @SuppressWarnings("unlikely-arg-type")
        @Override
        public void handle(MouseEvent e) {
            caption.setVisible(false);
            caption.setTranslateX(e.getSceneX());
            caption.setTranslateY(e.getSceneY() - 100);
            caption.setVisible(true);
        }
    });
    }
}
}

```

Добавена е и функционалност за промяна на скоростта на възпроизвеждане при изпращането на пакетите. Тя може да се промени от менюто Project -> Speed. За да бъдат коректни данните за час на пакетите, течението на времето се контролира от нишка и функционалността е дефинирана в класа „Singleton“.

Функция за натрупване на времето според избраната настройка от меню лентата:

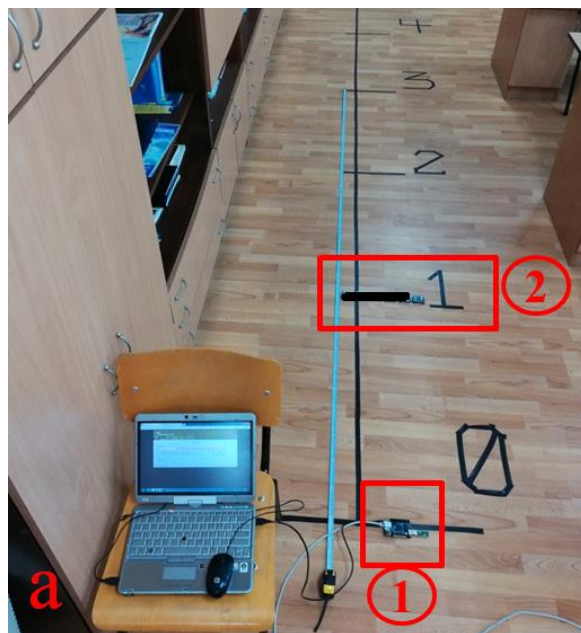
```
private void delayTime() {  
    ScheduledExecutorService exec =  
    Executors.newSingleThreadScheduledExecutor();  
    exec.scheduleAtFixedRate(new Runnable() {  
        @Override  
        public void run() {  
            switch (Singleton.getInstance().speed.toString()) {  
                case "1.0":  
                    date.setTime(date.getTime() + 1000);  
                    break;  
                case "0.75":  
                    date.setTime(date.getTime() + 750);  
                    break;  
                case "0.5":  
                    date.setTime(date.getTime() + 500);  
                    break;  
                case "0.25":  
                    date.setTime(date.getTime() + 250);  
                    break;  
  
                default:  
                    break;  
            }  
            ...  
        }  
    }  
}
```

Глава 6. Тестване на симулатора

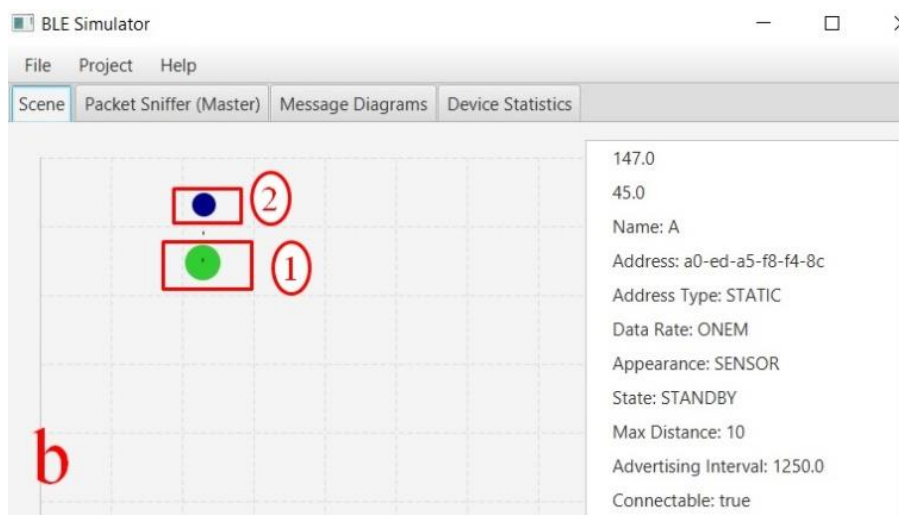
Извършено е идентично експериментално проучване за обменните съобщения при установяване на връзка, изпращане на данни и прекратяване на връзката между Master и Slave в реална (Фиг. 31) и симулирана (Фиг. 32) BLE мрежа. За да се осигури съпоставимост между резултатите от реалната и симулирана BLE мрежа, е реализирана експериментална топология на едно главно и едно подчинено устройство.

Реална BLE мрежа е изградена с Raspberri Pi 4 Model B [11] развойна среда с операционна система Raspbian. В развойната среда са инсталирани софтуерните пакети Bluetooth, Bluepy и Python, както и библиотеката libglib2.0-dev, които позволяват Raspberry да работи като BLE master устройство. Комуникацията с краен сензорен възел се извършва с помощта на вграден в RaspberryPi 4 платката, BLE приемо-предавател. Сензорният възел, с който е реализиран експеримента е CC2650STK [12] на Texas Instruments, който може да се конфигурира за работа с BLE технология.

Визуализирането на обменните съобщения при установяване на връзка, изпращане на данни и прекратяване на връзката между Master и Slave в BLE мрежа се извършва с прихващаните пакети през Wireshark програма за проследяване на трафика. Прихващането на пакети през Wireshark в BLE мрежа е възможно когато програмата се инсталира в операционната система на Raspberri Pi. За прихващане на пакети в BLE комуникационната среда, Wireshark прослушва трафика предаван през Bluetooth интерфейса на Raspberri Pi. Визуализирането на обменените съобщения между BLE Master и Slave е възможно от меню Statistics на Wireshark и избор на опцията Flow Graph.

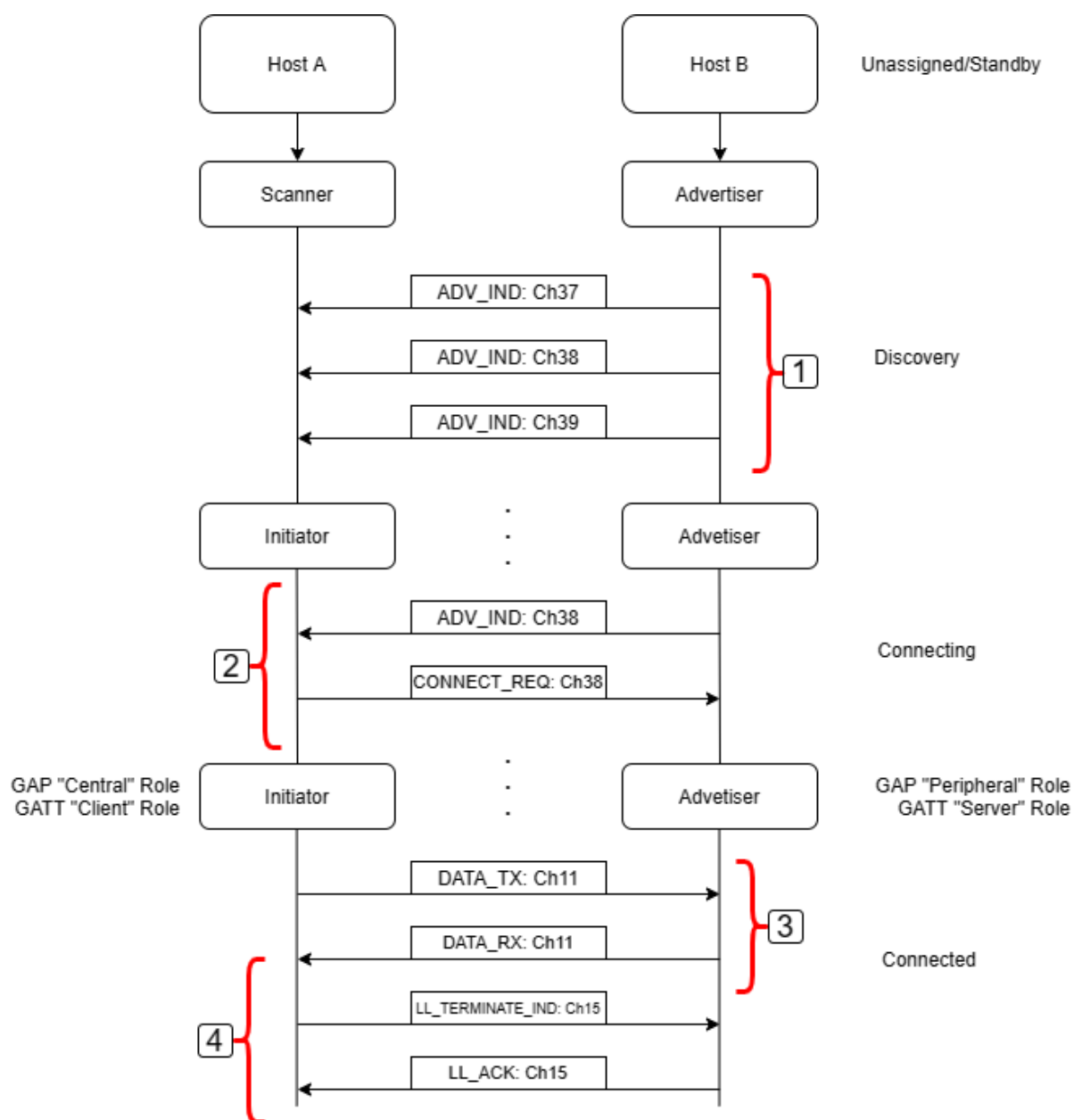


Фиг. 31. Експериментална топология при реална мрежа



Фиг. 32. Експериментална топология при симулирана мрежа

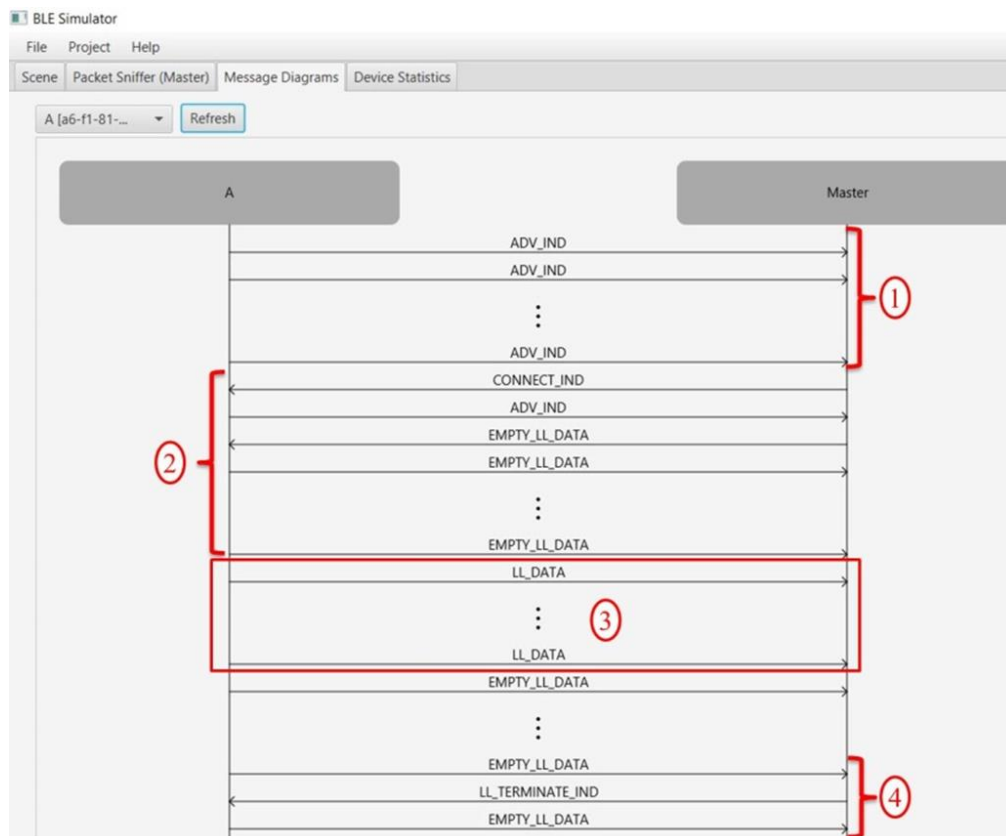
По стандарт една комуникация между главно и подчинено устройство, която включва основните видове комуникационни пакети, а именно такива за рекламиране, свързване и за изпращане на даннови пакети има вида показан на Фиг. 33. На Фиг. 34 е представена извършената комуникация между Master и Slave при реална BLE мрежа, а на Фиг. 35 при симулирана.



Фиг. 33. Комуникация между главно и подчинено устройство според BLE стандарта



Фиг. 34. Комуникация между главно и подчинено устройство в реална среда



Фиг. 35. Комуникация между главно и подчинено устройство в симулирана среда

По време на процеса на установяване на връзка, периферното устройство първоначално изпраща рекламни пакети през комуникационната среда, което позволява на централното устройство да го открие (Фиг. 33-1, Фиг. 34-1, Фиг. 35-1). Рекламните пакети могат да съдържат различни видове информация, но най-често включват име на устройство, мощност на предаване, UUID на услугите, които устройството поддържа. Понякога капацитетът на рекламните пакети не е достатъчен за изпращане на необходимите данни и в този случай могат да се използват заявка за сканиране и отговор за сканиране, които се считат за допълнителни рекламни данни.

След като централното устройство открие периферно в обхвата си, то може да се свърже с него, като само централно устройство може да инициира връзка, а периферното устройство да изпраща реклами и приема връзки. Когато централният реши да изгради връзка с периферното, той изпраща заявка за връзка към него. Тогава връзката се счита за изградена и централното устройство става Master, а периферното Slave. След изтичане на време, наречено Интервал на свързване, Master може да изпрати пакет данни на Slave. Подчиненият е длъжен да върне на главния пакет данни с информацията, необходима за комуникацията, когато получи пакета с данни от главния. Ако подчиненото устройство няма съответните данни, той изпраща празен пакет с данни на главния (Фиг. 33-2, Фиг. 34-2, Фиг. 35-2). При установена връзка, може да се извърши процес на трансфер на данни между Master и Slave (Фиг. 33-3, Фиг. 34-3, Фиг. 35-3). След приключване на изпращането на данни, връзката може да бъде прекратена чрез изпращане на "Disconnect (LL_TERMINATE_IND)" съобщение (Фиг. 33-3, Фиг. 34-4, Фиг. 35-4).

От проведения експеримент за реализиране на процеса на обмен на съобщения за установяване на връзка, изпращане на данни и прекратяване на връзката между Master и Slave според стандарта, при реална мрежа и симулация, съобщенията „Advertisement” и „Connection” са идентични, „Disconnect ” съобщението от реалната мрежа, се представя като „Terminate“ при симулация.

По стандарт и при симулацията няма допълнителни информационни съобщения по време на изграждането на връзката, видими в реална мрежа, тъй като тази информация се приема като част от изпратените реклами при симулация, а при обменяните съобщения по стандарт се представя принципът на работа. Освен това няма допълнителни информационни съобщения при предаване на данни и прекратяване на връзката при представяне на съобщенията по стандарт и при симулация.

Някои детайли на комуникацията по стандарт и при симулация са пропуснати. При представянето по стандарта е така, защото стандартът

специфицира правила, указания или дефиниции на характеристики за отговаряне на целта по предназначение, а при симулацията, за да се опрости разглежданият процес.

Глава 7. Заключение и изводи

Нарастващата необходимост от автоматизация и ускоряващият се растеж на пазара на интернет на нещата (IoT) влияят върху растежа на пазара на BLE от последните няколко години. Намира се все по-голямо приложение на устройствата, използващи тази технология. Нейните достоинства, а именно това, че консумира ниска мощност, нейната бързина на работа и изпращане на съобщенията, я отличават от останалите безжични услуги.

След обстоен анализ на съществуващите решения за симулиране или емулиране на BLE технологията се разбира, че съществуват малко такива, повечето от които изискват физически Bluetooth модули, за да може да се симулира работата на протокола.

Разработеният симулатор позволява изучаване на различни аспекти, свързани с работата на BLE стандарта, без да е необходимо наличието на физически устройства. Проведените експерименти показват, че симулаторът може да се използва за представяне на акцентите от комуникацията между главно и подчинено устройство.

Източници

- [1] **Mohammad Afaneh**. Intro to Bluetooth Low Energy. 2018, ISBN: 9781790198153.
- [2] **Silicon Labs**. UG103.14: Bluetooth® LE Fundamentals v0.7. <https://www.silabs.com/documents/public/user-guides/ug103-14-fundamentals-ble.pdf>. Дата на последно влизане 22.7.2021.
- [3] **Specification Working Group**. Bluetooth Core Specification v5.2. 2019.
- [4] **The pros and cons of Bluetooth Low Energy**. <https://www.electronicsworld.com/news/design/communications/pros-cons-bluetooth-low-energy-2014-10/>. Дата на последно влизане 22.7.2021.
- [5] **BLE Peripheral Simulator**. <https://github.com/WebBluetoothCG/ble-test-peripheral-android>. Дата на последно влизане 22.7.2021.
- [6] **Bluetooth Low Energy Communication MATLAB**. <https://www.mathworks.com/help/matlab/bluetooth-low-energy-communication.html>. Дата на последно влизане 22.7.2021.
- [7] **Introduction to Java**. <https://www.geeksforgeeks.org/introduction-to-java/>. Дата на последно влизане 22.7.2021.
- [8] **Apache Maven**. <https://maven.apache.org/>. Дата на последно влизане 22.7.2021
- [9] **JavaFX Documentation Project**. <https://fxdocs.github.io/docs/html5/>. Дата на последно влизане 22.7.2021
- [10] **Launch4j**. Cross-platform Java executable wrapper. <http://launch4j.sourceforge.net/>. Дата на последно влизане 21.7.2021
- [11] **Raspberry Pi 4 Tech Specs**. <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/specifications/>. Дата на последно влизане 22.07.2021.
- [12] **SimpleLink Multi-standard SensorTag**. <http://www.ti.com/tool/CC2650STK?keyMatch=CC2650STK&tisearch=Search-EN-everything&usecase=GPN>. Дата на последно влизане 22.07.2021.

Приложение 1. Терминологичен речник.

1M PHY: LE 1M е PHY, използван в Bluetooth 4. Той има скорост на обмен 1 Mbps. LE 1M продължава да се използва в Bluetooth 5. Неговата поддръжка е задължителна.

2M PHY: LE 2M PHY позволява на физическия слой да работи с 2 Mbps и по този начин позволява по-високи скорости на предаване на данни от LE 1M и Bluetooth 4. Поддръжката му не е задължителна.

ATT (Attribute Protocol): протокол за устройства за изброяване на атрибути, които позволяват различни операции като писане и четене.

BLE (Bluetooth Low Energy): безжична технология с малък обхват, фокусирана върху приложения с ниска мощност и ниска честотна лента.

CODED PHY: LE кодираният PHY позволява да се утрое обхватът (приблизително) в сравнение с Bluetooth без увеличаване на необходимата мощност на предаване, като за сметка на това се намалява скоростта на предаване (125 Kbps или 500Kbps).

FHSS (Frequency Hopping Spread Spectrum): метод за предаване на радиосигнали, чрез бързо превключване между различни честоти, като се използва алгоритъм за псевдослучайно избиране на честота, съгласуван от предавателя и приемника.

FXML: XML формат, който позволява съставянето на JavaFX GUI подобно на съставянето на уеб GUI в HTML. Позволява да се отдели кода на изгледа на JavaFX от останалия код на приложението.

GAP (General Access Profile): слой, отговорен за управлението на връзките, реклами, функции за откриване и сигурност.

GATT (General Attribute Profile): основен модел на данни, който позволява на устройствата да откриват, пишат, и прочетат елементи;

HCI (Host Controller Interface): стандартният протокол, който позволява на контролера да взаимодейства с хоста (в рамките на един и същ чипсет или в различни).

JDK (Java Development Kit): среда за разработка на софтуер, която се използва за разработване на Java приложения и аплети. Съдържа в себе си JRE и инструменти за разработка.

JRE (Java Runtime Environment): набор от софтуерни инструменти, които се използват за разработване на Java приложения. Използва се за осигуряване на среда за изпълнение. Съдържа набор от библиотеки, които JVM използва по време на изпълнение.

JVM (Java Virtual Machine): виртуална машина, която осигурява среда за изпълнение, в която може да бъде изпълнен байт кодът на Java. Той може също така да изпълнява тези програми, които са написани на други езици и компилирани в Java байт код.

L2CAP (Logical Link Control and Adaptation Control): слой в BLE архитектурата, който действа като протокол за мултиплексиране и обработка фрагментацията и рекомбинацията на пакети.

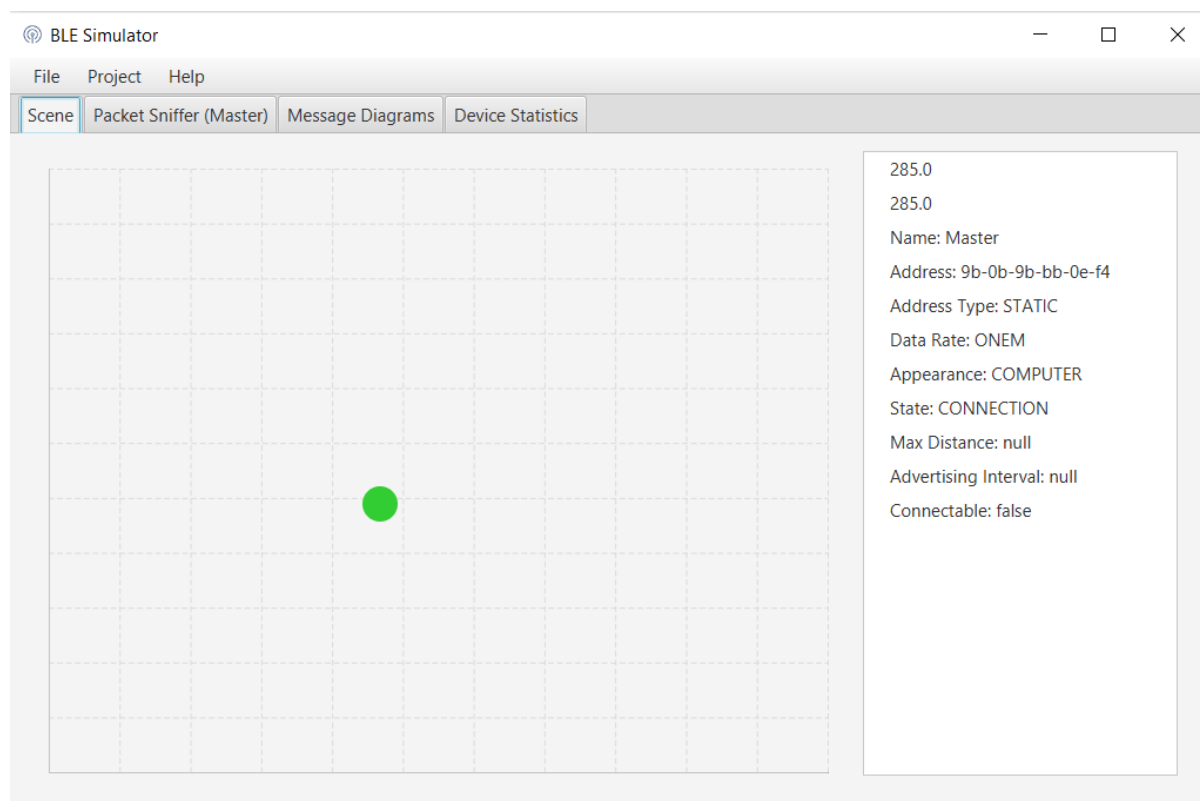
PDU (Protocol Data Unit): информация, която се предава като единична структура сред обекти в мрежа.

PHY (Physical Layer): слой, който представлява физическата верига, отговорна за предаване и приемане на радиопакети.

SM (Security Manager): слой в BLE архитектурата, който определя методите на сдвояване и разпределяне на ключове между две BLE устройства

Приложение 2. Ръководство на потребителя

За да се стартира симулатора, трябва да се отвори неговия .exe файл. Появява се началния екран на приложението, който е представен на Фиг. 1. На него са разположени двата основни елемента – меню лентата и лентата с табове, като всеки таб съдържа различна функционалност свързана с работата на BLE технологията.



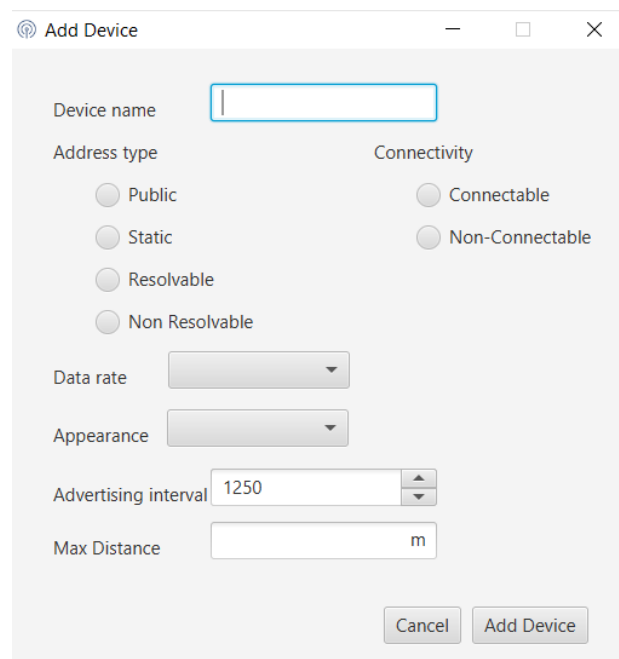
Фиг. 1. Начален екран на симулатора и табът Scene

Табът „Scene“ се разделя на две части:

- Област на устройствата (отляво), върху която се разполагат устройствата и могат да бъдат размествани, с цел да се симулира движение и промяна на обхвата. Главното устройство винаги присъства (устройството със зелен кръг);
- Информация за избраното устройство (отдясно), на което се изобразяват различни параметри на устройството, като координати върху областта на устройствата, име, адрес, изглед, и д.р.

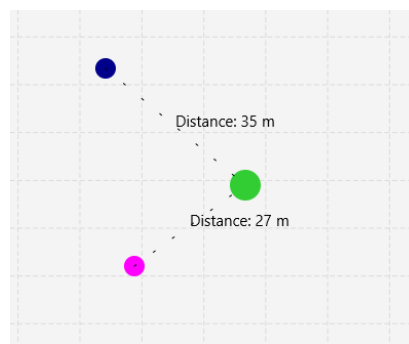
За да се добави устройство върху областта на устройствата, трябва от меню лентата да се избере „Project -> Add Device“ и ще се появи прозореца

показан на Фиг. 2. На него трябва да се въведат името на устройството, типа на адреса му, като когато е публичен, той трябва да се въведе ръчно, докато при другите се генерира автоматично, дали може да се свърже устройството или не, какъв е неговия data rate, изгледът му, който определя какъв вид услуги може да изпраща, интервала му на рекламиране и максималното разстояние, при което ще бъде в обхвата на главното устройство. Всички полета трябва да бъдат въведени, в противен случай няма да се позволи създаването на периферното устройство.



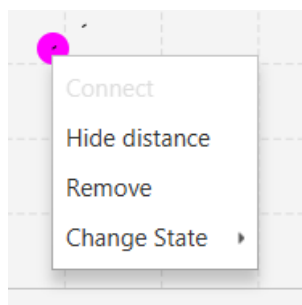
Фиг. 2. Прозорец за добавяне на периферни устройства

След добавянето на устройството, то ще се изобрази върху областта на устройствата, както е показано на Фиг. 3.



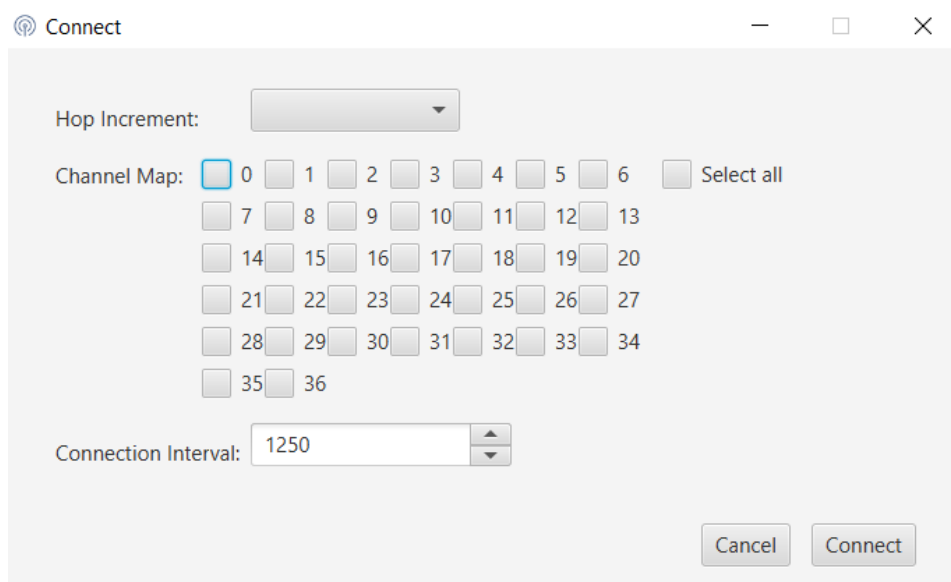
Фиг. 3. Изглед на няколко устройства в областта на устройствата

За допълнителни опции върху периферните устройства, с десен бутон се отваря контекстно меню, от което може да променяме състоянието на устройството от „Change State“, като опциите са „Advertising“ (рекламиране) и „Standby“ (в готовност), да скриваме или да показваме разстоянието от главното устройство и да премахнем устройството. При добавянето на устройство, неговото състояние по подразбиране е „в готовност“, ако го сменим на „рекламиране“, опцията „Connect“ (свързване), става достъпна.



Фиг. 4. Контекстно меню за периферните устройство

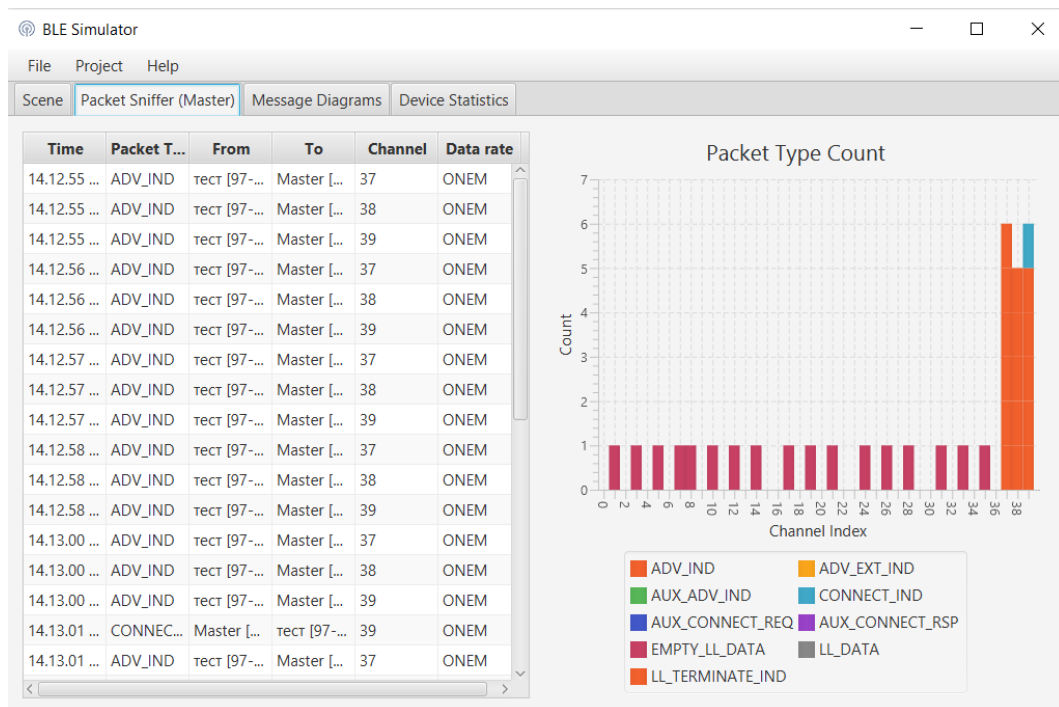
При избор на опцията за свързване, се появява прозорец за свързване, показан на Фиг. 5. При него трябва да зададем hop инкремента, който служи за изчисление за прескачането на каналите при връзка, мапинга на каналите, който показва, кои са позволените канали за комуникация, въпреки че алгоритъма може да позволи или забрани други, и интервала на свързване. Трябва всички полета да бъдат въведени, в противен случай, няма да се позволи свързването на периферното с главното устройство.



Фиг. 5. Прозорец за свързване

В табът „Packet Sniffer (Master)“ (Фиг. 6) може да се следят пакетите постъпили при и изпратени от главното устройство, както и да се следи диаграмата за бройка на различните типове пакети от таблицата. В таблицата с

пакети се показва информация за пакетите, като време на изпращане на пакета, типът му, от кого и за кого е, канала и data rate-а на устройството, което е изпратило пакета.



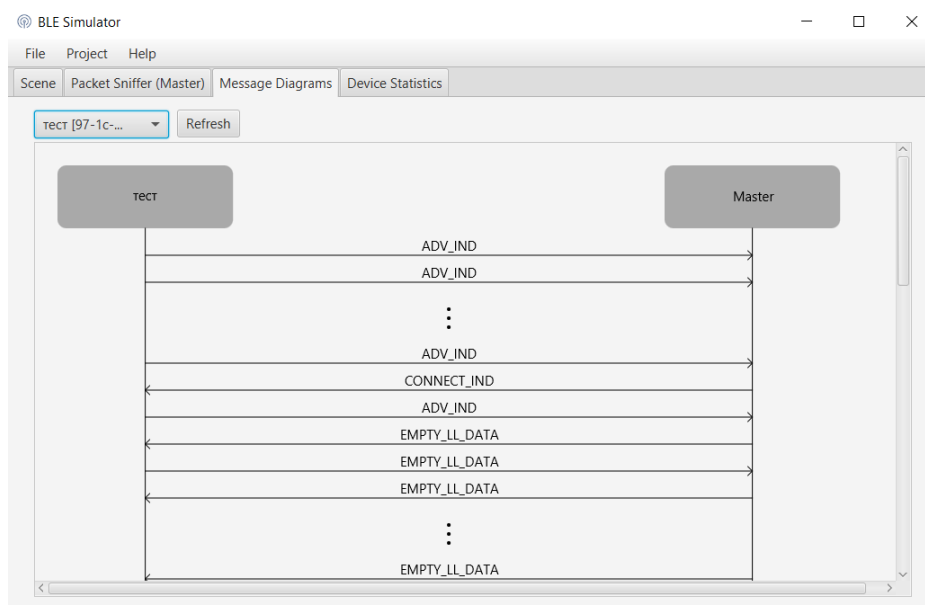
Фиг. 6. Табът „Packet Sniffer (Master)“

За да се провери допълнителна информация за пакета, от таблицата с двойно натискане на левия бутон ще се отвори съответния прозорец, който е показан на Фиг. 7.



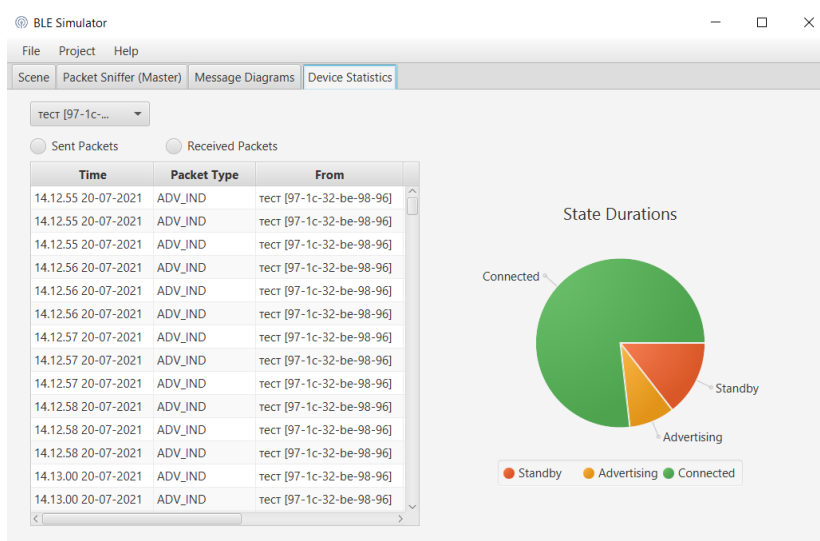
Фиг. 7. Прозорец с допълнителна информация за пакета

На третия таб („Message Diagrams“) се илюстрира поредицата от съобщения между главното устройство и избраното периферно. Това може да се види на Фиг. 8.



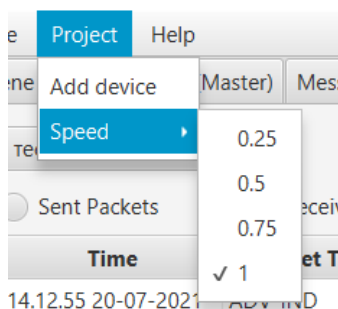
Фиг. 8. Диаграма за последователност на съобщенията

На последния таб („Device Statistics“) се намира таблица на изпратени и получени пакети и време на работа във всяко състояние за избрано периферно устройство. За да се види колко време в секунди е работата в дадено състояние трябва да се натисне с ляв бутон съответната му част от диаграмата и ще се изобрази.



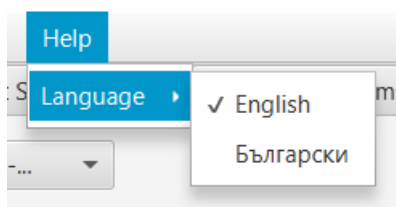
Фиг. 9. Табът „Device Statistics“

Добавена и функционалност за промяна на скоростта на симулацията, която е достъпна от меню лентата от „Project -> Speed“.



Фиг. 10. Опции за промяна на скоростта на симулатора

Възможна е и смяна на езика на потребителския интерфейс. Това става от меню лентата „Help -> Language“.



Фиг. 11. Опциите за смяна на език на потребителския интерфейс

Приложение 3. Код на приложението

model.Device:

```
public class Device {

    public enum State {
        STANDBY, ADVERTISING, SCANNING, INITIATING, CONNECTION, REMOVAL;
    }

    public enum Appearance {
        UNKNOWN, PHONE, COMPUTER, CLOCK, TAG, THERMOMETER, HEART_RATE_SENSOR,
BLOOD_PRESSURE, GLUCOSE_METER, SENSOR, OTHER
    }

    String name;
    DeviceAddress deviceAddress;
    State state;
    DataRate dataRate;
    Appearance appearance;
    DeviceCircle deviceCircle;
    ObservableList<Packet> packetsSent = FXCollections.observableArrayList();
    ObservableList<Packet> packetsReceived =
FXCollections.observableArrayList();
    ObservableList<Packet> allPackets = FXCollections.observableArrayList();
    DevicePacketFactory packetFactory = new DevicePacketFactory(this);
    LongProperty standbyTime = new SimpleLongProperty(0), advertisingTime = new
SimpleLongProperty(0),
        connectedTime = new SimpleLongProperty(0);

    public Device(String name, DeviceAddress deviceAddress, String state,
DataRate dataRate, Appearance appearance,
        DeviceCircle deviceCircle) {
        this.name = name;
        this.deviceAddress = deviceAddress;
        this.state = State.valueOf(state);
        this.dataRate = dataRate;
        this.appearance = appearance;
        this.deviceCircle = deviceCircle;
    }

    public Device() {
        this.name = "";
        this.deviceAddress = new DeviceAddress();
        this.state = State.REMOVAL;
        this.dataRate = DataRate.ERROR;
    }

    @Override
    public String toString() {
        return name + " [" + deviceAddress + "];"
    }

    public String getName() {
        return name;
    }

    public DeviceAddress getDeviceAddress() {
        return deviceAddress;
    }
}
```

```

}

public State getState() {
    return state;
}

public void setState(State state) {
    this.state = state;
}

public DataRate getDataRate() {
    return dataRate;
}

public Appearance getAppearance() {
    return appearance;
}

public DeviceCircle getDeviceCircle() {
    return deviceCircle;
}

public ObservableList<Packet> getPacketsSent() {
    return packetsSent;
}

public ObservableList<Packet> getPacketsReceived() {
    return packetsReceived;
}

public void addReceivedPacket(Packet packet) {
    this.packetsReceived.add(packet);
    this.allPackets.add(packet);
}

public void addSentPacket(Packet packet) {
    this.packetsSent.add(packet);
    this.allPackets.add(packet);
}

public DevicePacketFactory getPacketFactory() {
    return packetFactory;
}

public LongProperty getStandbyTime() {
    return standbyTime;
}

public void setStandbyTime(long standbyTime) {
    this.standbyTime.set(standbyTime);
}

public LongProperty getAdvertisingTime() {
    return advertisingTime;
}

public void setAdvertisingTime(long advertisingTime) {
    this.advertisingTime.set(advertisingTime);
}

```

```

    public LongProperty getConnectedTime() {
        return connectedTime;
    }

    public void setConnectedTime(long connectedTime) {
        this.connectedTime.set(connectedTime);
    }

    public ObservableList<Packet> getAllPackets() {
        return allPackets;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + ((dataRate == null) ? 0 :
dataRate.hashCode());
        result = prime * result + ((name == null) ? 0 : name.hashCode());
        result = prime * result + ((state == null) ? 0 : state.hashCode());
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Device other = (Device) obj;
        if (dataRate != other.dataRate)
            return false;
        if (name == null) {
            if (other.name != null)
                return false;
        } else if (!name.equals(other.name))
            return false;
        if (state != other.state)
            return false;
        return true;
    }

}

```

packets.Packet:

```

public class Packet {
    String time;
    PacketType packetType;
    Device deviceFrom;
    Device deviceTo;
    boolean status;
    String channel;
    DataRate dataRate;
    Packet secondaryPacket;
    String payload;
}

```



```

    public Packet(String time, PacketType packetType, Device deviceFrom, Device
deviceTo, String channel,
        DataRate dataRate) {
        this.time = time;
        this.packetType = packetType;
        this.deviceFrom = deviceFrom;
        this.deviceTo = deviceTo;
        this.status = false;
        this.channel = channel;
        this.dataRate = dataRate;
    }

    public Packet(String time, PacketType packetType, Device deviceFrom, Device
deviceTo, String channel,
        DataRate dataRate, String payload) {
        this.time = time;
        this.packetType = packetType;
        this.deviceFrom = deviceFrom;
        this.deviceTo = deviceTo;
        this.status = false;
        this.channel = channel;
        this.dataRate = dataRate;
        this.payload = payload;
    }

    public Packet(String time, PacketType packetType, Device deviceFrom, Device
deviceTo, String channel,
        DataRate dataRate, Packet secondaryPacket) {
        this.time = time;
        this.packetType = packetType;
        this.deviceFrom = deviceFrom;
        this.deviceTo = deviceTo;
        this.channel = channel;
        this.dataRate = dataRate;
        this.secondaryPacket = secondaryPacket;
    }

    public Packet(String time, PacketType packetType, Device deviceFrom, Device
deviceTo, String channel,
        DataRate dataRate, Packet secondaryPacket, String payload) {
        this.time = time;
        this.packetType = packetType;
        this.deviceFrom = deviceFrom;
        this.deviceTo = deviceTo;
        this.channel = channel;
        this.dataRate = dataRate;
        this.secondaryPacket = secondaryPacket;
        this.payload = payload;
    }

    public Packet() {
        this.time = "00.00.01 01-01-1999";
        this.packetType = PacketType.EMPTY_LL_DATA;
        this.deviceFrom = null;
        this.deviceTo = null;
        this.status = false;
        this.channel = "99";
    }

```

```

        this.dataRate = DataRate.ERROR;
        this.secondaryPacket = null;
    }

    public Packet(Packet o) {
        this.time = o.getTime();
        this.packetType = o.getPacketType();
        this.deviceFrom = o.getDeviceFrom();
        this.deviceTo = o.getDeviceTo();
        this.status = o.isStatus();
        this.channel = o.getChannel();
        this.dataRate = o.getDataRate();
        this.secondaryPacket = o.getSecondaryPacket();
        this.payload = o.getPayload();
    }

    public String getTime() {
        return time;
    }

    public void setTime(String time) {
        this.time = time;
    }

    public PacketType getPacketType() {
        return packetType;
    }

    public Device getDeviceFrom() {
        return deviceFrom;
    }

    public Device getDeviceTo() {
        return deviceTo;
    }

    public boolean isStatus() {
        return status;
    }

    public void setStatus(boolean status) {
        this.status = status;
    }

    public String getChannel() {
        return channel;
    }

    public void setChannel(String channel) {
        this.channel = channel;
    }

    public DataRate getDataRate() {
        return dataRate;
    }

    public Packet getSecondaryPacket() {
        return secondaryPacket;
    }

```

```

    public void setSecondaryPacket(Packet secondaryPacket) {
        this.secondaryPacket = secondaryPacket;
    }

    public String getPayload() {
        return payload;
    }

    public void setPayload(String payload) {
        this.payload = payload;
    }

    @Override
    public String toString() {
        return "Packet [time=" + time + ", packetType=" + packetType + ",
deviceFrom=" + deviceFrom + ", deviceTo="
            + deviceTo + ", status=" + status + ", channel=" +
channel + ", dataRate=" + dataRate
            + ", secondaryPacket=" + secondaryPacket + ", payload="
+ payload + "]\n";
    }

    public static void sendPacket(Packet packet) {
        packet.getDeviceFrom().addSentPacket(packet);
        packet.getDeviceTo().addReceivedPacket(packet);

        if (packet.getPacketType().equals(PacketType.ADV_EXT_IND)) {
            try {
                Thread.sleep(500);
                sendPacket(packet.getSecondaryPacket());
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }

    public boolean isValid() {
        if (this.dataRate.equals(DataRate.ERROR))
            return false;
        else
            return true;
    }

    public boolean isAdvertisingPacket() {
        Set<PacketType> advertisingPacketTypes =
EnumSet.of(PacketType.ADV_EXT_IND, PacketType.ADV_IND,
            PacketType.ADV_NONCONN_IND, PacketType.AUX_ADV_IND,
            PacketType.AUX_CONNECT_REQ,
            PacketType.AUX_CONNECT_RSP, PacketType.CONNECT_IND);

        return advertisingPacketTypes.contains(getPacketType());
    }

    public boolean isConnectionPacket() {
        return !isAdvertisingPacket();
    }

```

```

@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((channel == null) ? 0 :
channel.hashCode());
    result = prime * result + ((dataRate == null) ? 0 :
dataRate.hashCode());
    result = prime * result + ((deviceFrom == null) ? 0 :
deviceFrom.hashCode());
    result = prime * result + ((deviceTo == null) ? 0 :
deviceTo.hashCode());
    result = prime * result + ((packetType == null) ? 0 :
packetType.hashCode());
    result = prime * result + ((secondaryPacket == null) ? 0 :
secondaryPacket.hashCode());
    result = prime * result + (status ? 1231 : 1237);
    result = prime * result + ((time == null) ? 0 : time.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
    if (this == obj)
        return true;
    if (obj == null)
        return false;
    if (getClass() != obj.getClass())
        return false;
    Packet other = (Packet) obj;
    if (channel == null) {
        if (other.channel != null)
            return false;
    } else if (!channel.equals(other.channel))
        return false;
    if (dataRate != other.dataRate)
        return false;
    if (deviceFrom == null) {
        if (other.deviceFrom != null)
            return false;
    } else if (!deviceFrom.equals(other.deviceFrom))
        return false;
    if (deviceTo == null) {
        if (other.deviceTo != null)
            return false;
    } else if (!deviceTo.equals(other.deviceTo))
        return false;
    if (packetType != other.packetType)
        return false;
    if (secondaryPacket == null) {
        if (other.secondaryPacket != null)
            return false;
    } else if (!secondaryPacket.equals(other.secondaryPacket))
        return false;
    if (status != other.status)
        return false;
    if (time == null) {
        if (other.time != null)

```

```

        return false;
    } else if (!time.equals(other.time))
        return false;
    return true;
}
}

```

app.SuperMain:

```

public class SuperMain {

    public static void main(String[] args) {
        App.main(args);
    }

}

```

app.App:

```

public class App extends Application {

    @Override
    public void start(Stage stage) {

        Parent root;
        try {
            FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("/view/AppScene.fxml"));
            fxmlLoader.setResources(ResourceBundle.getBundle("view.lang",
new Locale("en")));
            root = fxmlLoader.load();

            stage.setScene(new Scene(root));
            stage.setTitle("BLE Simulator");
            stage.setMinWidth(root.minWidth(-1));
            stage.setMinHeight(root.minHeight(-1));
            stage.getIcons().add(new
Image(getClass().getResourceAsStream("/images/logo.png")));
            stage.show();
            stage.setOnHiding(new EventHandler<WindowEvent>() {

                @Override
                public void handle(WindowEvent event) {
                    Thread.currentThread().interrupt();
                    Singleton.getInstance().executor.shutdownNow();
                    Platform.exit();
                }
            });
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    public static void main(String[] args) {
        Launch();
    }
}

```

```
}
```

controller.AppController:

```
public class AppController implements Initializable {

    . . .

    ObservableList<Device> devices = Singleton.getInstance().devices;
    Device master = Singleton.getInstance().master;

    @Override
    public void initialize(URL arg0, ResourceBundle arg1) {
        addMaster();

        loadSpeedMenu();
        loadSlaveComboBox();
        loadMessageSequencePane();
        loadTableView();
        loadChannelBarChart();
        loadStatisticsTab();
        languageLoader();
    }

    private void languageLoader() {
        englishLanguageRadioButton.setSelected(true);

        ToggleGroup toggleGroup = new ToggleGroup();
        toggleGroup.getToggles().add(englishLanguageRadioButton);
        toggleGroup.getToggles().add(bulgarianLanguageRadioButton);
    }

    public void changeLanguageEnglish() {
        Locale.setDefault(new Locale("en"));
        loadView(new Locale("en"));
    }

    public void changeLanguageBulgarian() {
        Locale.setDefault(new Locale("bg"));
        loadView(new Locale("bg"));
    }

    private void loadView(Locale locale) {
        ResourceBundle bundle = ResourceBundle.getBundle("view.lang",
locale);

        fileMenuLabel.setText(bundle.getString("fileMenuLabel"));
        closeMenuLabel.setText(bundle.getString("closeMenuLabel"));
        projectMenuLabel.setText(bundle.getString("projectMenuLabel"));
        addDeviceMenuLabel.setText(bundle.getString("addDeviceMenuLabel"));
        speedMenuLabel.setText(bundle.getString("speedMenuLabel"));
        helpMenuLabel.setText(bundle.getString("helpMenuLabel"));
        languageMenuLabel.setText(bundle.getString("languageMenuLabel"));

        sceneTabLabel.setText(bundle.getString("sceneTabLabel"));
        snifferTabLabel.setText(bundle.getString("snifferTabLabel"));
        messageTabLabel.setText(bundle.getString("messageTabLabel"));
        statisticsTabLabel.setText(bundle.getString("statisticsTabLabel"));
    }
}
```

```

        channelStackedBarChart.setTitle(bundle.getString("packetChartLabel"));

        channelStackedBarChart.getYAxis().setLabel(bundle.getString("packetChartCou
ntLabel"));

        channelStackedBarChart.getXAxis().setLabel(bundle.getString("packetChartCha
nnelLabel"));

        refreshButton.setText(bundle.getString("refreshButton"));

        ratioPieChart.setTitle(bundle.getString("ratioPieChartLabel"));

        sentPacketsRadioButton.setText(bundle.getString("sentPacketsRadioButtonLabe
l"));

        receivedPacketsRadioButton.setText(bundle.getString("receivedPacketsRadioBu
ttonLabel"));

        loadTableView();
        DeviceStatisticsUtil.loadTableView();
        DeviceStatisticsUtil.loadPieChart();
    }

    private void toggleListener(ToggleGroup toggle) {
        toggle.selectedToggleProperty().addListener(new
ChangeListener<Toggle>() {
            @Override
            public void changed(ObservableValue<? extends Toggle> ov,
Toggle t, Toggle t1) {
                RadioButton selectedRadioButton = (RadioButton)
toggle.getSelectedToggle();

                if (selectedRadioButton.getText().equals("Sent
Packets"))

                    DeviceStatisticsUtil.setSentPacketsToggle(true);
                else
                    DeviceStatisticsUtil.setSentPacketsToggle(false);
            }
        });
    }

    private void loadMessageSequencePane() {
        MessageSequenceFactory.setMessageSequencePane(messageSequencePane);
        MessageSequenceFactory.setScrollPane(scrollPane);
    }

    public void refreshMessageDiagram(ActionEvent event) {
        if (MessageSequenceFactory.getCurrentSlave() != null)
            MessageSequenceFactory.refresh();
    }

    private void loadSlaveComboBox() {
        this.slaveComboBox.setItems(devices);
        this.slaveComboBox.valueProperty().addListener((ob, ov, nv) -> {
            MessageSequenceFactory.setCurrentSlave(nv);
        });
    }

```

```

        this.slaveInfoComboBox.setItems(devices);
        this.slaveInfoComboBox.valueProperty().addListener((ob, ov, nv) -> {
            if (nv != null)
                DeviceStatisticsUtil.setDevice(nv);
            else
                DeviceStatisticsUtil.setDevice(new Device());
        });
    }

    public void loadSpeedMenu() {
        speed1.setSelected(true);

        ToggleGroup toggleGroup = new ToggleGroup();
        toggleGroup.getToggles().add(speed025);
        toggleGroup.getToggles().add(speed05);
        toggleGroup.getToggles().add(speed075);
        toggleGroup.getToggles().add(speed1);
    }

    public void changeSpeed025() {
        changeSpeed("0.25");
    }

    public void changeSpeed05() {
        changeSpeed("0.5");
    }

    public void changeSpeed075() {
        changeSpeed("0.75");
    }

    public void changeSpeed1() {
        changeSpeed("1");
    }

    public void changeSpeed(String speed) {
        Singleton.getInstance().speed = Double.valueOf(speed);
    }

    private void loadStatisticsTab() {
        DeviceStatisticsUtil.setTableView(this.slavePacketTableView);
        DeviceStatisticsUtil.setPieChart(this.ratioPieChart);

        ToggleGroup deviceTableViewToggle = new ToggleGroup();
        this.sentPacketsRadioButton.setToggleGroup(deviceTableViewToggle);

        this.receivedPacketsRadioButton.setToggleGroup(deviceTableViewToggle);

        toggleListener(deviceTableViewToggle);
    }

    @SuppressWarnings("unchecked")
    public void loadChannelBarChart() {
        XYChart.Series<String, Integer> advInd = new XYChart.Series<>();
        advInd.setName("ADV_IND");

        XYChart.Series<String, Integer> advExtInd = new XYChart.Series<>();
        advExtInd.setName("ADV_EXT_IND");
    }

```



```

XYChart.Series<String, Integer> auxAdvInd = new XYChart.Series<>();
auxAdvInd.setName("AUX_ADV_IND");
. . .

XYChart.Series<String, Integer> terminateIND = new
XYChart.Series<>();
terminateIND.setName("LL_TERMINATE_IND");

master.getAllPackets().addListener(new ListChangeListener<Packet>() {

    @Override
    public void onChanged(Change<? extends Packet> c) {
        Packet lastPacket = c.getList().get(c.getList().size() -
1);

        Platform.runLater(() -> {
            switch (lastPacket.getPacketType()) {
                case ADV_IND:
                    advInd.getData().add(new
XYChart.Data<>(lastPacket.getChannel(), 1));
                    break;
                case ADV_EXT_IND:
                    advExtInd.getData().add(new
XYChart.Data<>(lastPacket.getChannel(), 1));
                    break;

                . . .

                    break;
                case LL_TERMINATE_IND:
                    terminateIND.getData().add(new
XYChart.Data<>(lastPacket.getChannel(), 1));
                    break;
                default:
                    break;
            }
        });
    }
});

for (int i = 0; i < 40; i++) {
    advExtInd.getData().add(new XYChart.Data<String,
Integer>(Integer.toString(i), 0));
}

this.channelStackedBarChart.setCategoryGap(0);
this.channelStackedBarChart.getData().setAll(advInd, advExtInd,
auxAdvInd, connectInd, auxConnectReq,
auxConnectRsp, emptyLLdata, llData, terminateIND);
}

@SuppressWarnings("unchecked")
public void loadTableView() {

    this.packetTableView.setRowFactory(tv -> {
        TableRow<Packet> row = new TableRow<>();
        row.setOnMouseClicked(event -> {
            if (event.getClickCount() == 2 && (!row.isEmpty())) {
                Parent root;

```

```

        try {
            PacketController.setPacket(row.getItem());
            root =
FXMLLoader.load(getClass().getResource("/view/PacketPane.fxml"));
            Stage stage = new Stage();
            stage.setTitle("Packet Info");
            stage.setScene(new Scene(root, 1000, 700));
            stage.getIcons().add(new
Image(getClass().getResourceAsStream("/images/logo.png")));
            stage.setResizable(false);
            stage.show();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    });
    return row;
});

    TableColumn<Packet, String> timeColumn = new
TableColumn<>("%timeLabel");
    timeColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("timeLabel"));
    timeColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("time"));
    timeColumn.setSortType(TableColumn.SortType.DESENDING);

    TableColumn<Packet, String> typeColumn = new
TableColumn<>("%typeLabel");
    typeColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("typeLabel"));
    typeColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("packetType"));

    . . .

    TableColumn<Packet, String> dataRateColumn = new
TableColumn<>("%dataRateLabel");
    dataRateColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("dataRateLabel"));
    dataRateColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("dataRate"));

    this.packetTableView.getColumns().setAll(timeColumn, typeColumn,
fromColumn, toColumn, channelColumn,
        dataRateColumn);

    this.packetTableView.setItems(master.getAllPackets());
}

    public void addDeviceButton(ActionEvent event) {
        Parent root;
        try {
            FXMLLoader fxmlLoader = new
FXMLLoader(getClass().getResource("/view/AddDeviceWindow.fxml"));
            fxmlLoader.setResources(ResourceBundle.getBundle("view.lang",
Locale.getDefault()));
            root = fxmlLoader.load();

```

```

        Stage stage = new Stage();
        stage.setTitle("Add Device");
        stage.getIcons().add(new
Image(getClass().getResourceAsStream("/images/logo.png")));
        stage.setScene(new Scene(root, 500, 500));
        stage.setResizable(false);
        stage.show();
        stage.setOnHiding(new EventHandler<WindowEvent>() {

            @Override
            public void handle(WindowEvent event) {
                if (AddDeviceController.getDevice() != null)
                    addDevice(AddDeviceController.getDevice());
            }
        });
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public void addDevice(Device device) {
    Singleton.getInstance().executor.execute(device.getPacketFactory());

    devices.add(device);
    ScenePaneUtil.makeDragable(device, scenePane);
    ScenePaneUtil.onClickListener(device, scenePane, detailsTreeView);

    this.scenePane.getChildren().add(device.getDeviceCircle().getCircle());
    this.scenePane.getChildren().add(device.getDeviceCircle().getLine());
    this.scenePane.getChildren().add(device.getDeviceCircle().getText());
    device.getDeviceCircle().getLine().setVisible(false);

    device.getDeviceCircle().getCircle().setLayoutX(100);
    device.getDeviceCircle().getCircle().setLayoutY(100);
}

private void addMaster() {
    ScenePaneUtil.makeDragable(master, scenePane);
    ScenePaneUtil.onClickListener(master, scenePane, detailsTreeView);

    this.scenePane.getChildren().add(master.getDeviceCircle().getCircle());
    master.getDeviceCircle().getCircle().setLayoutX(300);
    master.getDeviceCircle().getCircle().setLayoutY(300);

    MasterPacketFactory.listenForReceivedPacketsOnMaster(master);
}

public static String getRandomColorName() {
    Random random = new Random();
    String[] colors = { "DARKTURQUOISE", "AQUA", "DARKGREEN", "MAROON",
"GOLDENROD", "NAVY", "OLIVE", "DARKBLUE",
    "MAGENTA", "PERU" };

    return colors[random.nextInt(9)];
}

public void close() {
    Platform.exit();
}

```

```
}
```

controller.AddDeviceController:

```
public class AddDeviceController implements Initializable {

    AddressType selectedAddressType;
    DeviceAddress deviceAddress;
    static Device newDevice;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        loadDataRateChoiceBox();
        loadAppearanceComboBox();
        createToggleGroup();
        setUpIntervalSpinner();
    }

    private void setUpIntervalSpinner() {
        StringConverter<Double> doubleConverter = new
StringConverter<Double>() {
            private final DecimalFormat df = new DecimalFormat("###.###");

            @Override
            public String toString(Double object) {
                if (object == null) {
                    return "";
                }
                return df.format(object);
            }

            @Override
            public Double fromString(String string) {
                try {
                    if (string == null) {
                        return null;
                    }
                    string = string.trim();
                    if (string.length() < 1) {
                        return null;
                    }
                    return df.parse(string).doubleValue();
                } catch (ParseException ex) {
                    throw new RuntimeException(ex);
                }
            }
        };

        this.advertisingIntervalSpinner.getValueFactory().setConverter(doubleConver
ter);

        advertisingIntervalSpinner.focusedProperty().addListener((observable,
oldValue, newValue) -> {
            if (!newValue) {
                advertisingIntervalSpinner.increment(0);
            }
        });
    }
}
```

```

    }

    private void loadDataRateChoiceBox() {
        this.dataRateComboBox.setItems(
            FXCollections.observableArrayList(new DataRate[] {
                DataRate.ONEM, DataRate.TWOM, DataRate.CODED }));
    }

    private void loadAppearanceComboBox() {

        this.appearanceComboBox.setItems(FXCollections.observableArrayList(Device.A
            ppearance.values()));
    }

    private void createToggleGroup() {
        adresGroup = new ToggleGroup();
        publicRadioButton.setToggleGroup(adresGroup);
        staticRadioButton.setToggleGroup(adresGroup);
        resolvableRadioButton.setToggleGroup(adresGroup);
        nonResolvableRadioButton.setToggleGroup(adresGroup);

        connectableGroup = new ToggleGroup();
        connectableRadioButton.setToggleGroup(connectableGroup);
        nonConnectableRadioButton.setToggleGroup(connectableGroup);

        this.toggleListener();
    }

    private void toggleListener() {
        adresGroup.selectedToggleProperty().addListener(new
        ChangeListener<Toggle>() {
            @Override
            public void changed(ObservableValue<? extends Toggle> ov,
                Toggle t, Toggle t1) {
                RadioButton selectedRadioButton = (RadioButton)
                adresGroup.getSelectedToggle();
                String selectedValue = selectedRadioButton.getText();

                switch (selectedValue) {
                    case "Public":
                        publicAddressTextField.setVisible(true);
                        staticAddressLabel.setVisible(false);
                        resolvableAddressLabel.setVisible(false);
                        nonResolvableAddressLabel.setVisible(false);

                        selectedAddressType = AddressType.PUBLIC;
                        deviceAddress = new DeviceAddress("PUBLIC",
                            null);
                        break;
                    case "Static":
                        deviceAddress = new DeviceAddress("STATIC",
                            null);
                        staticAddressLabel.setText(deviceAddress.getAddress());

                        . . .

                        selectedAddressType = AddressType.STATIC;
                        break;
                }
            }
        });
    }

```

```

        case "Resolvable":
            . . .
            selectedAddressType = AddressType.RESOLVABLE;
            break;
        case "Non Resolvable":
            deviceAddress = new
DeviceAddress("NONRESOLVABLE", null);

            nonResolvableAddressLabel.setText(deviceAddress.getAddress());

            . . .

            selectedAddressType = AddressType.NONRESOLVABLE;
            break;
        }
    }
});
}

public void addDevice() {
    if (validateInput()) {
        errorLabel.setVisible(false);
        if (selectedAddressType != null) {
            if (selectedAddressType.equals(AddressType.PUBLIC))

                deviceAddress.setAddress(publicAddressTextField.getText());

                Circle circle = new Circle();
                circle.setRadius(10);
                circle.setId(deviceNameTextField.getText());

                circle.setFill(Color.valueOf(AppController.getRandomColorName()));

                DeviceCircle deviceCircle = new DeviceCircle(circle, new
Line());

                newDevice = new Device(deviceNameTextField.getText(),
deviceAddress, "STANDBY",

                dataRateComboBox.getSelectionModel().getSelectedItem(),

                appearanceComboBox.getSelectionModel().getSelectedItem(), deviceCircle);

                newDevice.getPacketFactory().setAdvertisingInterval(advertisingIntervalSpin
ner.getValue().toString());

                newDevice.getPacketFactory().setMaxDistance(maxDistanceTextField.getText())
;

                newDevice.getPacketFactory().setConnectable(
                    ((RadioButton)
connectableGroup.getSelectedToggle()).getText().equals("Connectable") ? true
                    : false);

                cancel();
            }
        } else {
            System.out.println("Input fields!");
            errorLabel.setVisible(true);

```

```

    }

    }

    private boolean validateInput() {
        return (InputValidator.isNotNull(deviceNameTextField) &&
InputValidator.isNotNull(dataRateComboBox)
                && InputValidator.isNotNull(appearanceComboBox) &&
InputValidator.textNumeric(maxDistanceTextField)
                &&
InputValidator.textNumericDouble(advertisingIntervalSpinner) &&
InputValidator.isNotNull(addressGroup)
                && InputValidator.isNotNull(connectableGroup));
    }

    public void cancel() {
        Stage stage = (Stage) staticAddressLabel.getScene().getWindow();
        stage.close();
        newDevice = null;
    }

    public static Device getDevice() {
        return newDevice;
    }
}

```

controller.ConnectionController:

```

public class ConnectionController implements Initializable {

    Runnable selectAllStateChangeProcessor;
    List<CheckBox> allCheckBoxes = new ArrayList<CheckBox>();
    static Device slave;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        populateComboBox();
        selectAllCheckBoxes();
        loadSpinner();
    }

    private void loadSpinner() {
        connIntervalSpinner.focusedProperty().addListener((observable,
oldValue, newValue) -> {
            if (!newValue) {
                connIntervalSpinner.increment(0);
            }
        });
    }

    private void populateComboBox() {
        for (int i = 5; i < 17; i++) {
            this.hopIncrementComboBox.getItems().add(Integer.toString(i));
        }
    }

    private void selectAllCheckBoxes() {
        allCheckBoxes = Arrays.asList(channel0, channel1, channel2,
channel3, channel4, channel5, channel6, channel7,

```

```

        channel8, channel9, channel10, channel11, channel12,
channel13, channel14, channel15, channel16,
        channel17, channel18, channel19, channel20, channel21,
channel22, channel23, channel24, channel25,
        channel26, channel27, channel28, channel29, channel30,
channel31, channel32, channel33, channel34,
        channel35, channel36);

        allCheckBoxes.forEach(box ->
box.selectedProperty().addListener((observable, wasSelected, isSelected) -> {
            if (selectAllStateChangeProcessor == null) {
                boolean allSelected =
allCheckBoxes.stream().map(CheckBox::isSelected).reduce(true, (a, b) -> a && b);

                boolean anySelected =
allCheckBoxes.stream().map(CheckBox::isSelected).reduce(false, (a, b) -> a || b);

                if (allSelected) {
                    this.selectAllCheckBox.setSelected(true);
                    this.selectAllCheckBox.setIndeterminate(false);
                }

                if (!anySelected) {
                    this.selectAllCheckBox.setSelected(false);
                    this.selectAllCheckBox.setIndeterminate(false);
                }

                if (anySelected && !allSelected) {
                    this.selectAllCheckBox.setSelected(false);
                    this.selectAllCheckBox.setIndeterminate(true);
                }
            }
        }));

        this.selectAllCheckBox.setAllowIndeterminate(true);

        this.selectAllCheckBox.selectedProperty()
            .addListener((observable, wasSelected, isSelected) ->
scheduleSelectAllStateChangeProcessing());
        this.selectAllCheckBox.indeterminateProperty().addListener(
            (observable, wasIndeterminate, isIndeterminate) ->
scheduleSelectAllStateChangeProcessing());
    }

    private void scheduleSelectAllStateChangeProcessing() {
        if (selectAllStateChangeProcessor == null) {
            selectAllStateChangeProcessor =
this::processSelectAllStateChange;
            Platform.runLater(selectAllStateChangeProcessor);
        }
    }

    private void processSelectAllStateChange() {
        if (!this.selectAllCheckBox.isIndeterminate()) {
            allCheckBoxes.forEach(box ->
box.setSelected(this.selectAllCheckBox.isSelected()));
        }
        selectAllStateChangeProcessor = null;
    }
}

```



```

    public void connect() {
        if (validateInput()) {
            errorLabel.setVisible(false);
            slave.getPacketFactory().setConnectionController(this);
            if
(ConnectionUtil.startConnectionEvent(Singleton.getInstance().master, slave,
getChannelMap()))

                try {

                    cancel();

                    Singleton.getInstance().executor.submit(new
Runnable() {

                        @Override
                        public void run() {
                            try {

                                ConnectionUtil.sendEmptyDataPacket(Singleton.getInstance().master, slave,
                                getChannelMap(),
                                getHopIncrement(),

                                String.valueOf(Math.ceil(

                                Double.parseDouble(slave.getPacketFactory().getAdvertisingInterval())

                                + connIntervalSpinner.getValue())));
                            } catch (NumberFormatException |
InterruptedException e) {
                                e.printStackTrace();
                            }
                        }

                    });

                } catch (NumberFormatException | InterruptedException e)
{
                    e.printStackTrace();
                }
            } else {
                System.out.println("Input fields!");
                errorLabel.setVisible(true);
            }
        }

        private boolean validateInput() {
            return (getChannelMap().size() > 0 &&
InputValidator.isNotNull(hopIncrementComboBox)
&&
InputValidator.textNumericDouble(connIntervalSpinner));
        }

        public void cancel() throws InterruptedException {
            Stage stage = (Stage) connectButton.getScene().getWindow();
            stage.close();
        }
    }

```

```

    public List<String> getChannelMap() {
        return allCheckBoxes.stream().filter(ch ->
ch.isSelected()).map(CheckBox::getText).collect(Collectors.toList());
    }

    public String getHopIncrement() {
        return this.hopIncrementComboBox.getValue();
    }

    public String getConnectionInterval() {
        return this.connIntervalSpinner.getValue().toString();
    }

    public static void setSlave(Device s) {
        slave = s;
    }
}

```

controller.SendPacketController:

```

public class SendPacketController implements Initializable {

    static Device slave;

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        loadView();
    }

    private void loadView() {
        loadAppearance();
        loadServiceCombo();
        loadSpinner();
        loadCheckBox();
    }

    private void loadAppearance() {
        appearanceLabel.setText(slave.getAppearance().toString());
    }

    private void loadServiceCombo() {
        switch (slave.getAppearance()) {
            case BLOOD_PRESSURE:
                serviceComboBox.setItems(FXCollections.observableArrayList(
                    new Services[] { Services.DEVICE_INFORMATION,
Services.BLOOD_PRESSURE, Services.USER_DATA }));
                break;
            case CLOCK:
                serviceComboBox.setItems(FXCollections.observableArrayList(
                    new Services[] { Services.CURRENT_TIME,
Services.DEVICE_INFORMATION, Services.USER_DATA }));
                break;
            . . .
            case UNKNOWN:
                serviceComboBox.setItems(FXCollections.observableArrayList(

```

```

        new Services[] { Services.CURRENT_TIME,
Services.GLUCOSE, Services.DEVICE_INFORMATION,
        Services.HEARTH_RATE,
Services.PHONE_ALERT_STATUS, Services.BLOOD_PRESSURE,
        Services.LOCATION_AND_NAVIGATION,
Services.PULSE_OXYMETER, Services.USER_DATA }));
        break;
    default:
        break;
    }
}

private void loadSpinner() {
    quantitySpinner.focusedProperty().addListener((observable, oldValue,
newValue) -> {
        if (!newValue) {
            quantitySpinner.increment(0);
        }
    });
}

private void loadCheckBox() {
    noneRadio.setToggleGroup(permissionsGroup);
    readableRadio.setToggleGroup(permissionsGroup);
    writableRadio.setToggleGroup(permissionsGroup);
    readableAndWritableRadio.setToggleGroup(permissionsGroup);
}

private void generateDataPackets() {
    String data = "Attribute Handle:" + "\nAttribute Type:Primary
Service" + "\nAttribute Value:"
        + serviceComboBox.getValue() + "\nAttribute
Permissions:"
        + ((RadioButton)
permissionsGroup.getSelectedToggle()).getText();

    Packet dataPacket = new Packet(Singleton.getTime(),
PacketType.LL_DATA, slave, Singleton.getInstance().master,

    ConnectionUtil.nextChannel(slave.getPacketFactory().getConnectionController
()).getChannelMap(),

    slave.getPacketFactory().getConnectionController().getHopIncrement()),
        slave.getDataRate(), data);

    slave.getPacketFactory().setSendDataPackets(dataPacket,
quantitySpinner.getValue().intValue());
}

public void send() {
    if (validateInput()) {
        errorLabel.setVisible(false);
        if (!slave.getPacketFactory().isHavingPacketsToSend()) {
            generateDataPackets();
            cancel();
        } else {
            System.out.println("Still has packets to send!");
        }
    }
}

```

```

        } else {
            errorLabel.setVisible(true);
        }
    }

    public void cancel() {
        Stage stage = (Stage) appearancelabel.getScene().getWindow();
        stage.close();
    }

    private boolean validateInput() {
        return (InputValidator.isNotNull(serviceComboBox) &&
            InputValidator.textNumericInteger(quantitySpinner)
            && InputValidator.isNotNull(permissionsGroup));
    }

    public static void setSlave(Device s) {
        System.out.println(s);
        slave = s;
    }
}

```

controller.PacketController:

```

public class PacketController implements Initializable {

    static Packet packet;
    ObservableList<Rectangle> rectangles = FXCollections.observableArrayList();

    @Override
    public void initialize(URL location, ResourceBundle resources) {
        draw();

        if (packet.getPayload() != null)
            loadTreeView();
    }

    public static void setPacket(Packet pckt) {
        packet = pckt;
    }

    private void loadTreeView() {
        TreeItem<String> rootItem = new TreeItem<String>("Payload Data");
        rootItem.setExpanded(true);
        addTreeViewData(packet.getPayload(), rootItem);

        this.payloadTreeView.setRoot(rootItem);
        AnchorPane.setTopAnchor(this.payloadTreeView, 400.0);
    }

    private void addTreeViewData(String payload, TreeItem<String> rootItem) {

        List<String> items = Arrays.asList(payload.split("\\r?\\n"));
        items.forEach(i -> {
            List<String> currentItem = Arrays.asList(i.split(":"));
            System.out.println(currentItem.get(0));
            TreeItem<String> label = new
TreeItem<String>(currentItem.get(0));

```

```

        label.setExpanded(true);

        if (currentItem.size() > 1) {
            List<String> subItems =
Arrays.asList(currentItem.get(1).split(", "));
            subItems.forEach(sI -> {

                if (currentItem.get(0).equals("Local name")) {
                    System.out.println(sI.getBytes().length);

                } else if (currentItem.get(0).equals("Advertising
interval")) {

                    System.out.println(Long

                        .toBinaryString(Double.doubleToLongBits((Double.parseDouble(sI) /
0.625))).length());

                }

                label.getChildren().add(new
TreeItem<String>(sI));
            });

            rootItem.getChildren().add(label);
        }
    });

}

private void draw() {
    int initialX = 35, initialY = 100;
    addRectangle(initialX, initialY, "Preamble", 65, 65, 1);
    addRectangle(getNextInitialX(initialX), initialY, "Access Address",
160, 65, 4);
    Rectangle payload = addRectangle(getNextInitialX(initialX), initialY,
"Protocol Data Unit (PDU)", 500, 65, 0);

    addRectangleSize(generatePDU(initialX, initialY), payload);

    addRectangle(getNextInitialX(initialX), initialY, "CRC", 130, 65, 3);
}

private int generatePDU(int initialX, int initialY) {
    int sizeValue = 0;
    Rectangle previousRectangle = drawNextLines();
    initialX = previousRectangle.xProperty().intValue();
    initialY =
previousRectangle.yProperty().add(previousRectangle.heightProperty().intValue() +
30);

    switch (packet.getPacketType()) {
        case ADV_EXT_IND:
            addRectangle((initialX), initialY, "Header", 125, 65, 2);
            Rectangle AdvExtPayload =
addRectangle(getNextInitialX(initialX), initialY, "Advertising Payload", 375, 65,
0);

            initialX = previousRectangle.xProperty().intValue();
            initialY = (int) initialY + 30 + 65;
    }
}

```

```

        sizeValue = generatePayload(initialX, initialY);
        addRectangleSize(sizeValue, AdvExtPayload);

        sizeValue += 2;
        break;
    case ADV_IND:
    case ADV_NONCONN_IND:
        addRectangle((initialX), initialY, "Header", 125, 65, 2);
        Rectangle AdvOneMpayload =
addRectangle(getNextInitialX(initialX), initialY, "Advertising Payload", 375, 65,
              0);

        initialX = previousRectangle.xProperty().intValue();
        initialY = (int) initialY + 30 + 65;

        sizeValue = generatePayload(initialX, initialY);
        addRectangleSize(sizeValue, AdvOneMpayload);

        sizeValue += 2;
        break;

    . . .

    case LL_DATA:
        addRectangle((initialX), initialY, "Header", 125, 65, 2);
        Rectangle dataPayload =
addRectangle(getNextInitialX(initialX), initialY, "Data Payload", 250, 65, 0);

        initialX = previousRectangle.xProperty().intValue();
        initialY = (int) initialY + 30 + 65;
        sizeValue = generatePayload(initialX, initialY);
        addRectangleSize(sizeValue, dataPayload);

        initialY =
previousRectangle.yProperty().add(previousRectangle.heightProperty().intValue() +
30;
        addRectangle(getNextInitialX(initialX), initialY, "MIC", 125,
65, 4);

        sizeValue += 6;
        break;
    default:
        break;
}

return sizeValue;
}

private int generatePayload(int initialX, int initialY) {
    int sizeValue = 0;

    Rectangle previousRectangle = drawNextLines();
    initialX = previousRectangle.xProperty().intValue();
    initialY =
previousRectangle.yProperty().add(previousRectangle.heightProperty().intValue() +
30;

    switch (packet.getPacketType()) {
    case ADV_EXT_IND:

```

```

        addRectangle((initialX), initialY, "Adv Mode", 75, 65, 1);

        if (packet.getDeviceFrom().getPacketFactory().isConnectable())
        {
            addRectangle(getNextInitialX(initialX), initialY,
"AdvDataInfo", 150, 65, 2);
            addRectangle(getNextInitialX(initialX), initialY, "Aux
Ptr", 150, 65, 3);
            sizeValue = 1 + 2 + 3;
        } else {
            addRectangle(getNextInitialX(initialX), initialY,
"AdvA", 300, 65, 6);
            sizeValue = 1 + 6;
        }

        break;
    case ADV_IND:
    case ADV_NONCONN_IND:
        int payloadSizeOneM =
packet.getDeviceFrom().getName().getBytes().length
+
(packet.getDeviceFrom().getPacketFactory().isConnectable() ? 2 : 0) + 5 + Long
        .toBinaryString(Double.doubleToLongBits((Double.parseDouble(
        packet.getDeviceFrom().getPacketFactory().getAdvertisingInterval()) /
0.625)))
        .length() / 8;

        addRectangle((initialX), initialY, "AdvA", 150, 65, 6);
        addRectangle(getNextInitialX(initialX), initialY, "AdvData",
225, 65, payloadSizeOneM);

        sizeValue = payloadSizeOneM + 6;
        break;

        . . .

    case LL_DATA:
        int payloadSizeData = 2 + 2 + 2 + 1 + 12;

        addRectangle((initialX), initialY, "L2CAP Hdr", 100, 65, 4);
        addRectangle(getNextInitialX(initialX), initialY, "OpCode",
50, 65, 1);
        addRectangle(getNextInitialX(initialX), initialY, "Data", 100,
65, payloadSizeData);

        sizeValue = payloadSizeData + 4 + 1;
        break;
    default:
        break;
}

return sizeValue;
}

private Rectangle drawNextLines() {
    Rectangle payloadRectangle = rectangles.get(rectangles.size() - 1);
    int startX = payloadRectangle.xProperty().intValue();

```

```

        int endX =
payloadRectangle.xProperty().add(payloadRectangle.widthProperty()).intValue();
        int y =
payloadRectangle.yProperty().add(payloadRectangle.heightProperty()).intValue();

        Line line1 = new Line(startX, y, startX, y + 30);
        Line line2 = new Line(endX, y, endX, y + 30);

        anchorPane.getChildren().addAll(line1, line2);

        return payloadRectangle;
    }

    private Rectangle addRectangle(int startX, int startY, String label, int
width, int height, int biteSize) {
        Rectangle rectangle = new Rectangle(startX, startY, width, height);
        rectangle.setFill(Color.DARKGRAY);

        rectangle.setStyle(
            "-fx-background-color: black, red; -fx-background-
insets: 0, 5; -fx-min-width: 20; -fx-min-height:20; -fx-max-width:20; -fx-max-
height: 20;");

        addBorder(rectangle);

        anchorPane.getChildren().add(rectangle);
        rectangles.add(rectangle);

        addRectangleTitle(label, rectangle);

        if (biteSize != 0)
            addRectangleSize(biteSize, rectangle);

        return rectangle;
    }

    private void addBorder(Rectangle rectangle) {
        Line leftBorder = new Line(rectangle.xProperty().doubleValue(),
rectangle.yProperty().add(1).doubleValue(),
            rectangle.xProperty().doubleValue(),

            rectangle.yProperty().add(rectangle.heightProperty()).subtract(1).doubleVal
ue());

        Line rightBorder = new
Line(rectangle.xProperty().add(rectangle.widthProperty()).doubleValue(),
            rectangle.yProperty().doubleValue(),
            rectangle.xProperty().add(rectangle.widthProperty()).doubleValue(),

            rectangle.yProperty().add(rectangle.heightProperty()).doubleValue());

        Line topBorder = new Line(rectangle.xProperty().doubleValue(),
rectangle.yProperty().doubleValue(),

            rectangle.xProperty().add(rectangle.widthProperty()).doubleValue(),
            rectangle.yProperty().doubleValue());

        Line bottomBorder = new Line(rectangle.xProperty().doubleValue(),

```



```

        rectangle.yProperty().add(rectangle.heightProperty()).doubleValue(),
        rectangle.xProperty().add(rectangle.widthProperty()).doubleValue(),
        rectangle.yProperty().add(rectangle.heightProperty()).doubleValue());

        anchorPane.getChildren().addAll(leftBorder, rightBorder, topBorder,
bottomBorder);
    }

    private void addRectangleTitle(String label, Rectangle rectangle) {
        Text text = new Text(label);
        text.setBoundsType(TextBoundsType.VISUAL);

        text.xProperty().bind((rectangle.xProperty()).add(rectangle.widthProperty()
.divide(2))
            .subtract(text.getLayoutBounds().getWidth() / 2));
        text.yProperty().bind((rectangle.yProperty()).subtract(10));

        anchorPane.getChildren().addAll(text);
    }

    private void addRectangleSize(int biteSize, Rectangle rect) {
        Text text = new Text(String.valueOf(biteSize));
        text.setBoundsType(TextBoundsType.VISUAL);
        text.xProperty().bind(
            (rect.xProperty()).add(rect.widthProperty().divide(2)).subtract(text.getLay
outBounds().getWidth() / 2));
        text.yProperty().bind((rect.yProperty()).add(rect.heightProperty().divide(2
)));

        anchorPane.getChildren().addAll(text);
    }

    private int getNextInitialX(int initialX) {
        return rectangles.get(rectangles.size() - 1).xProperty()
            .add(rectangles.get(rectangles.size() -
1).widthProperty()).intValue();
    }
}

```

utilities.Singleton:

```

public class Singleton {
    private static Singleton single_instance = null;

    public Device master;
    public ObservableList<Device> devices;
    public ThreadPoolExecutor executor;
    public Double speed = 1.0;
    public static Date date = new Date();
    private int currentSecond = (int) (date.getTime() % 10000 / 1000);

    private Singleton() {
        devices = FXCollections.observableArrayList();
        executor = (ThreadPoolExecutor) Executors.newCachedThreadPool();
    }
}

```

```

        createMaster();
        delayTime();
    }

    private void createMaster() {
        Circle circle = new Circle();
        circle.setRadius(15);
        circle.setFill(Color.LIMEGREEN);
        circle.setId("Master");
        DeviceCircle deviceCircle = new DeviceCircle(circle, new Line());

        master = new Device("Master", new DeviceAddress("STATIC", null),
"CONNECTION", DataRate.ONEM,
            Device.Appearance.COMPUTER, deviceCircle);
    }

    public static Singleton getInstance() {
        if (single_instance == null)
            single_instance = new Singleton();

        return single_instance;
    }

    public static String getTime() {
        SimpleDateFormat formatter = new SimpleDateFormat("HH.mm.ss dd-MM-
yyyy");
        return formatter.format(date);
    }

    public static String getAddedTime(int addedMilliseconds) {
        SimpleDateFormat formatter = new SimpleDateFormat("HH.mm.ss dd-MM-
yyyy");
        return formatter.format(date.getTime() + addedMilliseconds);
    }

    private void delayTime() {

        ScheduledExecutorService exec =
Executors.newSingleThreadScheduledExecutor();
        exec.scheduleAtFixedRate(new Runnable() {
            @Override
            public void run() {
                // System.out.println("tick");
                switch (Singleton.getInstance().speed.toString()) {
                    case "1.0":
                        date.setTime(date.getTime() + 1000);
                        break;
                    case "0.75":
                        date.setTime(date.getTime() + 750);
                        break;
                    case "0.5":
                        date.setTime(date.getTime() + 500);
                        break;
                    case "0.25":
                        date.setTime(date.getTime() + 250);
                        break;

                    default:
                        break;
                }
            }
        }, 0, 1, TimeUnit.SECONDS);
    }

```

```

        }

        if ((date.getTime() % 10000 / 1000) != currentSecond) {
            currentSecond = (int) (date.getTime() % 10000 /
1000);

            addStateDurations();
        }

    }

    }, 0, 1, TimeUnit.SECONDS);
}

private void addStateDurations() {
    devices.forEach(d -> {
        switch (d.getState()) {
            case ADVERTISING:
                d.setAdvertisingTime(d.getAdvertisingTime().add(1).longValue());
                break;
            case CONNECTION:
                d.setConnectedeTime(d.getConnectedeTime().add(1).longValue());
                break;
            case STANDBY:
                d.setStandbyTime(d.getStandbyTime().add(1).longValue());
                break;
            default:
                break;
        }
    });
}
}
}

```

utilities.MessageSequenceFactory:

```

public class MessageSequenceFactory {
    static AnchorPane messageSequencePane;
    static Device currentSlave;
    static ScrollPane scrollPane;
    static int numberOfMessages = 1;
    static boolean drewDot = false;

    public AnchorPane getMessageSequencePane() {
        return messageSequencePane;
    }

    public static void setMessageSequencePane(AnchorPane messageSqncPane) {
        messageSequencePane = messageSqncPane;
    }

    public static void setScrollPane(ScrollPane scr1Pane) {
        scrollPane = scr1Pane;
    }

    public static Device getCurrentSlave() {
        return currentSlave;
    }

    public static void setCurrentSlave(Device slave) {

```

```

        currentSlave = slave;
        refresh();
    }

    private static void enableResize() {

        messageSequencePane.prefHeightProperty().bind(scrollPane.heightProperty());

        messageSequencePane.prefWidthProperty().bind(scrollPane.widthProperty());
    }

    public static void refresh() {
        enableResize();
        clear();
        draw();
    }

    private static void draw() {
        if (currentSlave != null)
            makeDeviceRectangle(25, 25, currentSlave.getName());
        else
            makeDeviceRectangle(25, 25, "No device selected");
        makeDeviceRectangle(700, 25, "Master");

        ObservableList<Packet> packets = FXCollections

        .observableArrayList(Singleton.getInstance().master.getAllPackets().stream(
)
                                .filter(p ->
p.getDeviceFrom().equals(currentSlave) || p.getDeviceTo().equals(currentSlave))
                                .collect(Collectors.toList()));

        for (int i = 0; i < packets.size(); i++) {

            if
            (packets.get(i).getPacketType().equals(PacketType.ADV_EXT_IND)
             ||
            packets.get(i).getPacketType().equals(PacketType.AUX_ADV_IND)) {

                if (i > 2 &&
                packets.get(i).getPacketType().equals(packets.get(i - 2).getPacketType())
                &&
                packets.get(i).getDeviceFrom().equals(packets.get(i - 2).getDeviceFrom())) {
                    if (!drawedDot) {
                        drawRepeatingDots();
                        drawedDot = true;
                        generateMessageLines(packets.get(i - 1));
                        generateMessageLines(packets.get(i));
                    }
                } else {
                    generateMessageLines(packets.get(i));
                    drawedDot = false;
                }
            } else if
            (packets.get(i).getPacketType().equals(PacketType.EMPTY_LL_DATA)) {
                if (packets.get(i).getPacketType().equals(packets.get(i
- 1).getPacketType()))

```

```

        &&
packets.get(i).getPacketType().equals(packets.get(i - 2).getPacketType())
        &&
packets.get(i).getPacketType().equals(packets.get(i - 3).getPacketType())
        &&
packets.get(i).getDeviceFrom().equals(packets.get(i - 2).getDeviceFrom())
        && packets.get(i -
1).getDeviceFrom().equals(packets.get(i - 3).getDeviceFrom())) {
            if (!drawedDot) {
                drawRepeatingDots();
                drawedDot = true;
                generateMessageLines(packets.get(i - 1));
                generateMessageLines(packets.get(i));
            }
        } else {
            generateMessageLines(packets.get(i));
            drawedDot = false;
        }
    } else {
        if (i > 1 &&
packets.get(i).getPacketType().equals(packets.get(i - 1).getPacketType())
        &&
packets.get(i).getDeviceFrom().equals(packets.get(i - 1).getDeviceFrom())) {
            if (!drawedDot) {
                drawRepeatingDots();
                drawedDot = true;
                generateMessageLines(packets.get(i));
            }
        }
        else {
            generateMessageLines(packets.get(i));
            drawedDot = false;
        }
    }
}

}

private static void clear() {
    messageSequencePane.getChildren().clear();
    numberOfMessages = 1;
}

private static void makeDeviceRectangle(int startX, int startY, String
deviceName) {
    Rectangle device = new Rectangle(startX, startY, 200, 70);
    device.setArcHeight(20);
    device.setArcWidth(20);
    device.setFill(Color.DARKGRAY);
    device.maxHeight(70);
    device.maxWidth(600);

    device.widthProperty().bind(messageSequencePane.widthProperty().multiply(0.
2));

```

```

        device.setId(deviceName + "Rectangle");

        Line deviceLine = new Line(device.getX() + (device.getWidth() / 2),
            device.getY() + device.getHeight(),
            device.getX() + (device.getWidth() / 2),
            messageSequencePane.getHeight());

        deviceLine.startXProperty().bind(device.xProperty().add(device.widthProperty().divide(2)));
        deviceLine.startYProperty().bind(device.yProperty().add(device.heightProperty()));

        deviceLine.endXProperty().bind(deviceLine.startXProperty()); // works

        deviceLine.endYProperty().bind(messageSequencePane.heightProperty().add(30));

        deviceLine.setId(deviceName + "Line");

        Text text = new Text(deviceName);
        text.setBoundsType(TextBoundsType.VISUAL);

        text.xProperty().bind((device.xProperty().add(device.widthProperty().divide(2))
            .subtract(text.getLayoutBounds().getWidth() / 2));

        text.yProperty().bind((device.yProperty().add(device.heightProperty().divide(2))
            .add(text.getLayoutBounds().getHeight() / 2));

        messageSequencePane.getChildren().addAll(device, deviceLine, text);
    }

    private static void generateMessageLines(Packet packet) {
        Rectangle deviceFromRectangle =
            packet.getDeviceFrom().equals(currentSlave)
                ? (Rectangle) messageSequencePane.lookup("#" +
                    packet.getDeviceFrom().getName() + "Rectangle")
                : (Rectangle)
                    messageSequencePane.lookup("#MasterRectangle");

        Rectangle deviceToRectangle =
            packet.getDeviceTo().equals(currentSlave)
                ? (Rectangle) messageSequencePane.lookup("#" +
                    packet.getDeviceTo().getName() + "Rectangle")
                : (Rectangle)
                    messageSequencePane.lookup("#MasterRectangle");

        Line deviceLine = packet.getDeviceFrom().equals(currentSlave)
            ? (Line) messageSequencePane.lookup("#" +
                packet.getDeviceFrom().getName() + "Line")
            : (Line) messageSequencePane.lookup("#MasterLine");

        Line deviceLineTo = packet.getDeviceTo().equals(currentSlave)
            ? (Line) messageSequencePane.lookup("#" +
                packet.getDeviceTo().getName() + "Line")
            : (Line) messageSequencePane.lookup("#MasterLine");
    }

```

```

        Arrow messageArrow = new Arrow(deviceLine.getStartX(),
            deviceFromRectangle.getLayoutY() +
deviceFromRectangle.getHeight() + 15, deviceLineTo.getEndX(),
            deviceToRectangle.getLayoutY() +
deviceToRectangle.getHeight() + 15,
            packet.getDeviceFrom().equals(currentSlave) ?
Arrow.Direction.RIGHT : Arrow.Direction.LEFT);

        messageSequencePane.getChildren().addAll(messageArrow.getLine1(),
messageArrow.getLine2(),
            messageArrow.getLine());

        messageArrow.getLine().startXProperty().bind(deviceLine.startXProperty());

        messageArrow.getLine().endXProperty().bind(deviceLineTo.startXProperty());

        messageArrow.getLine().startYProperty()

            .bind(deviceFromRectangle.heightProperty().add(deviceFromRectangle.getY()).
add(30 * numberOfMessages));
        messageArrow.getLine().endYProperty()

            .bind(deviceFromRectangle.heightProperty().add(deviceFromRectangle.getY()).
add(30 * numberOfMessages));

        Text text = new Text(packet.getPacketType().toString());
        text.setBoundsType(TextBoundsType.VISUAL);

        text.xProperty().bind(messageArrow.getLine().startXProperty().add(messageAr
row.getLine().endXProperty()
            .divide(2).subtract(text.getLayoutBounds().getWidth() /
2));

        text.yProperty().bind(messageArrow.getLine().startYProperty().subtract(5));

        messageSequencePane.getChildren().add(text);

        numberOfMessages++;
    }

    private static void drawRepeatingDots() {
        Line masterLine = (Line) messageSequencePane.lookup("#MasterLine");
        Line slaveLine = (Line) messageSequencePane.lookup("#" +
currentSlave.getName() + "Line");
        Circle c1 = new Circle(2);

        c1.centerXProperty().bind(slaveLine.startXProperty().add(masterLine.startXP
roperty()).divide(2));
        c1.centerYProperty().bind(masterLine.startYProperty().add(30 *
numberOfMessages));

        Circle c2 = new Circle(2);
        c2.centerXProperty().bind(c1.centerXProperty());
        c2.centerYProperty().bind(c1.centerYProperty().add(10));

        Circle c3 = new Circle(2);
        c3.centerXProperty().bind(c2.centerXProperty());

```

```

        c3.centerYProperty().bind(c2.centerYProperty().add(10));

        numberOfMessages += 2;

        messageSequencePane.getChildren().addAll(c1, c2, c3);
    }
}

class Arrow {
    enum Direction {
        LEFT, RIGHT;
    }

    Line line;
    Line arrowLine1 = new Line();
    Line arrowLine2 = new Line();

    public Arrow(double startX, double startY, double endX, double endY,
        Direction direction) {
        line = new Line(startX, startY, endX, endY);

        if (direction.equals(Direction.RIGHT)) {

            arrowLine1.startXProperty().bind(line.endXProperty().subtract(5));

            arrowLine1.startYProperty().bind(line.endYProperty().subtract(5));

            arrowLine1.endXProperty().bind(line.endXProperty());
            arrowLine1.endYProperty().bind(line.endYProperty());

            arrowLine2.startXProperty().bind(line.endXProperty().subtract(5));
            arrowLine2.startYProperty().bind(line.endYProperty().add(5));

            arrowLine2.endXProperty().bind(line.endXProperty());
            arrowLine2.endYProperty().bind(line.endYProperty());
        } else {
            arrowLine1.startXProperty().bind(line.endXProperty());
            arrowLine1.startYProperty().bind(line.endYProperty());

            arrowLine1.endXProperty().bind(line.endXProperty().add(5));

            arrowLine1.endYProperty().bind(line.endYProperty().subtract(5));

            arrowLine2.startXProperty().bind(line.endXProperty());
            arrowLine2.startYProperty().bind(line.endYProperty());

            arrowLine2.endXProperty().bind(line.endXProperty().add(5));
            arrowLine2.endYProperty().bind(line.endYProperty().add(5));
        }
    }

    public Line getLine() {
        return this.line;
    }
}

```



```

    public Line getLine1() {
        return arrowLine1;
    }

    public Line getLine2() {
        return arrowLine2;
    }
}

```

utilities.DevicePacketFactory:

```

public class DevicePacketFactory implements Runnable {
    Device device;
    String currentChannel = "37";
    String secondaryChannel = Integer.toString(new Random().nextInt(37));
    private final Object lock = new Object();
    ConnectionController connectionController;
    String maxDistance;
    String advertisingInterval;
    boolean connectable, havingPacketsToSend;
    Packet packetToSend;
    int packetToSendCount;

    public DevicePacketFactory(Device device) {
        super();
        this.device = device;
    }

    @Override
    public void run() {
        synchronized (lock) {
            try {
                switch (device.getState()) {
                    case ADVERTISING:
                        System.out.println("adv");
                        advertisementEvent();
                        this.run();
                        break;
                    case CONNECTION:
                        System.out.println("conn");
                        connectionEvent();
                        this.run();
                        break;
                    case STANDBY:
                        System.out.println("standby");
                        lock.wait();
                        this.run();
                        break;
                    case REMOVAL:
                        System.out.println("removal");
                        break;

                    default:
                        break;
                }
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}

```

```

    }

}

private void advertisementEvent() throws InterruptedException {
    advertise();
    listenForConnectionRequest();

    Thread.sleep((long) (10 / Singleton.getInstance().speed));

    swapAdvertisingChannel();
}

public void advertise() throws InterruptedException {
    Packet extendAdvertisementPacket = null;

    if (this.device.getState().equals(Device.State.ADVERTISING)) {
        String advData = isConnectable() ? "Local name:" +
device.getName()
            + "\nFlags:LE Limited Discoverable Mode,LE
General Discoverable Mode,BR/EDR Not Supported,Simultaneous LE and BR/EDR
(Controller),Simultaneous LE and BR/EDR (Host)"
            + "\nAppearance:" + device.getAppearance() +
"\nLE Bluetooth Device Address: "
            + device.getDeviceAddress().getAddress() +
"\nAdvertising interval:" + advertisingInterval
            : "Local name:" + device.getName() +
"\nAppearance:" + device.getAppearance()
            + "\nLE Bluetooth Device Address:" +
device.getDeviceAddress().getAddress()
            + "\nAdvertising interval:" +
advertisingInterval;

        Packet advertisementPacket = new Packet(
            this.device.getDataRate().equals(DataRate.ONEM) ?
Singleton.getTime() : Singleton.getAddedTime(30),
            this.device.getDataRate().equals(DataRate.ONEM) ?
PacketType.ADV_IND : PacketType.AUX_ADV_IND,
            this.device, Singleton.getInstance().master,
            this.device.getDataRate().equals(DataRate.ONEM) ?
currentChannel : secondaryChannel,
            this.device.getDataRate(), advData);

        if (!this.device.getDataRate().equals(DataRate.ONEM)) {
            String extData = "Advertisement secondary channel:" +
secondaryChannel + "\nPHY:" + device.getDataRate()
                + "\nTime for auxiliary packet:" +
Singleton.getAddedTime(30);

            extendAdvertisementPacket = new
Packet(Singleton.getTime(), PacketType.ADV_EXT_IND, this.device,
Singleton.getInstance().master,
currentChannel, this.device.getDataRate(), advertisementPacket,
extData);
        }

        if (device.getDeviceCircle().getLine().isVisible()) {
            Packet.sendPacket(extendAdvertisementPacket == null ?
advertisementPacket : extendAdvertisementPacket);
        }
    }
}

```

```

        } else
            device.addSentPacket(extendAdvertisementPacket == null ?
advertisementPacket : extendAdvertisementPacket);
    }

    private void swapAdvertisingChannel() throws InterruptedException {
        switch (this.currentChannel) {
            case "37":
                this.currentChannel = "38";
                break;
            case "38":
                this.currentChannel = "39";
                break;
            case "39":
                this.currentChannel = "37";
                Thread.sleep(
                    (long)
(Long.valueOf(this.advertisingInterval.substring(0,
this.advertisingInterval.indexOf(".")))
                    / Singleton.getInstance().speed));
                break;
        }
    }

    private void listenForConnectionRequest() {
        if (!this.device.getState().equals(Device.State.CONNECTION)) {
            Packet connectionRequestPacket =
device.getPacketsReceived().stream()
                .filter(p -> p.getPacketType()

                .equals(this.device.getDataRate().equals(DataRate.ONEM) ?
PacketType.CONNECT_IND
                :
PacketType.AUX_CONNECT_REQ)
                && !p.isStatus())
                .reduce((first, second) -> second).orElse(new
Packet());

            if (connectionRequestPacket.getDeviceFrom() != null) {
                if
(!connectionRequestPacket.getDataRate().equals(DataRate.ONEM)) {
                    String connRspData = "Advertiser Address:" +
device.getDeviceAddress().getAddress()
                        + "Target Address:" +
Singleton.getInstance().master.getDeviceAddress().getAddress();

                    Packet connectionResponsePacket = new
Packet(Singleton.getTime(), PacketType.AUX_CONNECT_RSP,
                        this.device,
Singleton.getInstance().master, secondaryChannel, this.device.getDataRate(),
                        connRspData);

                    Packet.sendPacket(connectionResponsePacket);
                }

                connectionRequestPacket.setStatus(true);
                this.device.setState(Device.State.CONNECTION);
            }
        }
    }

```

```

        }
    }
}

private void connectionEvent() {
    try {
        Thread.sleep(Double.valueOf(connectionController.getConnectionInterval()).longValue());
        listenForReceivedPackets();
        dataPacketFactory();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

private void dataPacketFactory() {
    if (this.device.getState().equals(Device.State.CONNECTION)) {
        if (isHavingPacketsToSend()) {
            sendDataPacket();
        } else {
            returnEmptyDataPacket();
        }
    }
}

public void returnEmptyDataPacket() {
    if (!isHavingPacketsToSend()) {
        try {
            ConnectionUtil.sendEmptyDataPacket(device,
Singleton.getInstance().master,
this.connectionController.getChannelMap(),
this.connectionController.getHopIncrement(),
this.connectionController.getConnectionInterval());
        } catch (NumberFormatException | InterruptedException e) {
            e.printStackTrace();
        }
    }
}

public void setSendDataPackets(Packet dataPacket, int count) {
    setHavingPacketsToSend(true);
    packetToSend = dataPacket;
    packetToSendCount = count;
}

private void sendDataPacket() {
    if (packetToSendCount < 1)
        setHavingPacketsToSend(false);
    else {
        Packet buffPacket = new Packet(packetToSend);
        Packet.sendPacket(buffPacket);
        packetToSendCount--;
        System.out.println(packetToSendCount);
    }
}

```

```

        packetToSend.setChannel(ConnectionUtil.nextChannel(getConnectionController(
).getChannelMap(),
                                getConnectionController().getHopIncrement()));
    }

    }

    private void listenForReceivedPackets() {
        Packet lastReceivedPacket =
this.device.getPacketsReceived().get(this.device.getPacketsReceived().size() - 1);
        if (!lastReceivedPacket.isStatus())
            switch (lastReceivedPacket.getPacketType()) {
                case EMPTY_LL_DATA:
                    lastReceivedPacket.setStatus(true);
                    break;
                case LL_TERMINATE_IND:
                    lastReceivedPacket.setStatus(true);
                    this.device.setState(Device.State.STANDBY);
                    break;
                default:
                    break;
            }
    }

    }

    public void setMaxDistance(String maxDistance) {
        this.maxDistance = maxDistance;
    }

    public void setAdvertisingInterval(String advertisingInterval) {
        this.advertisingInterval = advertisingInterval;
    }

    public void setConnectionController(ConnectionController
connectionController) {
        this.connectionController = connectionController;
    }

    public String getMaxDistance() {
        return maxDistance;
    }

    public Object getLock() {
        return lock;
    }

    public boolean isConnectable() {
        return connectable;
    }

    public void setConnectable(boolean connectable) {
        this.connectable = connectable;
    }

    public boolean isHavingPacketsToSend() {
        return havingPacketsToSend;
    }

    public void setHavingPacketsToSend(boolean havingPacketsToSend) {

```

```

        this.havingPacketsToSend = havingPacketsToSend;
    }

    public String getAdvertisingInterval() {
        return advertisingInterval;
    }

    public ConnectionController getConnectionController() {
        return connectionController;
    }
}

```

utilities.ConnectionUtil:

```

public class ConnectionUtil {
    static String lastUnmappedChannel = "0";

    public static Packet checkForAdvertisementPacket(Device master, Device
slave) throws ParseException {
        Packet receivedOneMAdvertisement =
master.getPacketsReceived().stream().filter(
            p -> (p.getPacketType().equals(PacketType.ADV_IND) &&
p.getDeviceFrom().equals(slave) && !p.isStatus()))
            .reduce((first, second) -> second).orElse(new Packet());

        Packet receivedExtendedAdvertisement = master
            .getPacketsReceived().stream().filter(p ->
(p.getPacketType().equals(PacketType.ADV_EXT_IND)
            && p.getDeviceFrom().equals(slave) &&
!p.isStatus()))
            .reduce((first, second) -> second).orElse(new Packet());

        SimpleDateFormat formatter = new SimpleDateFormat("HH.mm.ss dd-MM-
yyyy");

        if (receivedOneMAdvertisement != null || receivedExtendedAdvertisement
!= null)
            return formatter.parse(receivedOneMAdvertisement.getTime())

                .after(formatter.parse(receivedExtendedAdvertisement.getTime())) ?
receivedOneMAdvertisement
                    : receivedExtendedAdvertisement;

        return null;
    }

    public static Packet checkForConnectionResponse(Device master, Device
slave) {
        Packet receivedOneMAdvertisement = slave
            .getPacketsReceived().stream().filter(p ->
(p.getPacketType().equals(PacketType.CONNECT_IND)
            && p.getDeviceFrom().equals(master) &&
!p.isStatus()))
            .reduce((first, second) -> second).orElse(new Packet());

        Packet receivedTwoMCodedAdvertisement = master
            .getPacketsReceived().stream().filter(p ->
(p.getPacketType().equals(PacketType.AUX_CONNECT_RSP)

```

```

        && p.getDeviceFrom().equals(slave) &&
!p.isStatus()))
        .reduce((first, second) -> second).orElse(new Packet());

        return receivedOneMAdvertisement.isValid() ? receivedOneMAdvertisement
: receivedTwoMCodedAdvertisement;
    }

    public static boolean startConnectionEvent(Device master, Device slave,
List<String> chMap) {
        if (slave.getState().equals(Device.State.AVERTISING)) {
            try {
                Packet lastReceivedPacket =
checkForAdvertisementPacket(master, slave);
                if (lastReceivedPacket.getDataRate() != DataRate.ERROR)
{
                    lastReceivedPacket.setStatus(true);

                    String llData = "Access Address:" + "\nCrc Init:"
+ "\nWindow Size:" + "\nWindow Offset:"
+ "\nInterval:" +
slave.getPacketFactory().getConnectionController().getConnectionInterval()
+ "\nLatency:" + "\nTimeout:" +
"\nChannel Map:";

                    for (int i = 0; i < chMap.size(); i++) {
                        llData = llData.concat(chMap.get(i));
                        if (i < chMap.size() - 1) {
                            llData = llData.concat(",");
                        }
                    }

                    llData = llData
                        .concat("\nHop:" +
slave.getPacketFactory().getConnectionController().getHopIncrement()
+ "\nSleep Clock
Accuracy:");

                    Packet connPacket = new
Packet(Singleton.getTime(),

                        lastReceivedPacket.getDataRate().equals(DataRate.ONEM) ?
PacketType.CONNECT_IND
:
PacketType.AUX_CONNECT_REQ,
                        master, slave,

                        lastReceivedPacket.getDataRate().equals(DataRate.ONEM) ?
lastReceivedPacket.getChannel()
:
lastReceivedPacket.getSecondaryPacket().getChannel(),
                        lastReceivedPacket.getDataRate(),
                        llData);

                    Packet.sendPacket(connPacket);

                    return true;
                }
            }
        }
    }

```

```

        } catch (ParseException e) {
            e.printStackTrace();
        }

        return false;
    }

    return false;
}

public static boolean sendEmptyDataPacket(Device from, Device to,
List<String> channelMap, String hopIncrement,
String connectionInterval) throws NumberFormatException,
InterruptedException {

    if (Thread.currentThread().getName() != "JavaFX Application Thread")
        Thread.sleep(Double.valueOf(connectionInterval).longValue());

    Device slave = from.getName().equals("Master") ? to : from;
    if (slave.getState().equals(Device.State.CONNECTION)) {

        String currentChannel = nextChannel(channelMap, hopIncrement);

        String emptyData = "Info:Empty PDU";
        Packet emptyDataPacket = new Packet(Singleton.getTime(),
PacketType.EMPTY_LL_DATA, from, to, currentChannel,
            slave.getDataRate(), emptyData);

        if (!from.getName().equals("Master")) {
            if (from.getDeviceCircle().getLine().isVisible()) {
                Packet.sendPacket(emptyDataPacket);
            } else
                from.addSentPacket(emptyDataPacket);
        } else {
            if (to.getDeviceCircle().getLine().isVisible()) {
                Packet.sendPacket(emptyDataPacket);
            } else
                from.addSentPacket(emptyDataPacket);
        }

        return true;
    } else
        return false;
}

public static String nextChannel(List<String> channelMap, String
hopIncrement) {
    int unmappedChannel = (Integer.valueOf>LastUnmappedChannel) +
Integer.valueOf(hopIncrement)) % 37;
    LastUnmappedChannel = Integer.toString(unmappedChannel);

    if (channelMap.contains(Integer.toString(unmappedChannel))) {
        return Integer.toString(unmappedChannel);
    } else {
        return Integer.toString(unmappedChannel % channelMap.size());
    }
}

```



```

    }
}

```

utilities.DeviceStatisticsUtil:

```

public class DeviceStatisticsUtil {
    static Device device;
    static TableView<Packet> tableView;
    static PieChart pieChart;
    static boolean sentPacketsToggle = true;
    static Label caption = new Label("");

    public static Device getDevice() {
        return device;
    }

    public static void setDevice(Device device) {
        DeviceStatisticsUtil.device = device;
        Load();
    }

    public static TableView<Packet> getTableView() {
        return tableView;
    }

    public static void setTableView(TableView<Packet> tableView) {
        DeviceStatisticsUtil.tableView = tableView;
    }

    public static PieChart getPieChart() {
        return pieChart;
    }

    public static void setPieChart(PieChart pieChart) {
        DeviceStatisticsUtil.pieChart = pieChart;
    }

    public static boolean isSentPacketsToggle() {
        return sentPacketsToggle;
    }

    public static void setSentPacketsToggle(boolean sentPacketsToggle) {
        DeviceStatisticsUtil.sentPacketsToggle = sentPacketsToggle;
        LoadTableView();
    }

    public static void load() {
        AnchorPane parent = (AnchorPane) pieChart.getParent().getParent();
        parent.getChildren().remove(caption);

        caption.setVisible(false);

        LoadTableView();

        parent.getChildren().add(caption);
        LoadPieChart();
    }
}

```

```

    public static void loadTableView() {
        if (sentPacketsToggle)
            LoadTable(device.getPacketsSent());
        else
            LoadTable(device.getPacketsReceived());
    }

    @SuppressWarnings("unchecked")
    public static void loadTable(ObservableList<Packet> packetList) {
        tableView.setRowFactory(tv -> {
            TableRow<Packet> row = new TableRow<>();
            row.setOnMouseClicked(event -> {
                if (event.getClickCount() == 2 && (!row.isEmpty())) {
                    Parent root;
                    try {
                        PacketController.setPacket(row.getItem());
                        root =
FXMLLoader.Load(DeviceStatisticsUtil.class.getResource("/view/PackagePane.fxml"));
                        Stage stage = new Stage();
                        stage.setTitle("Packet Info");
                        stage.setScene(new Scene(root, 1000, 700));
                        stage.getIcons().add(new
Image(DeviceStatisticsUtil.class.getResourceAsStream("/images/logo.png")));
                        stage.setResizable(false);
                        stage.show();
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                }
            });
        });
        return row;
    });

    TableColumn<Packet, String> timeColumn = new
TableColumn<>("%timeLabel");
    timeColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("timeLabel"));
    timeColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("time"));
    timeColumn.setSortType(TableColumn.SortType.DECENDING);

    TableColumn<Packet, String> typeColumn = new
TableColumn<>("%typeLabel");
    typeColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("typeLabel"));
    typeColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("packetType"));

    TableColumn<Packet, String> fromColumn = new
TableColumn<>("%fromLabel");
    fromColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("fromLabel"));
    fromColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("deviceFrom"));

    TableColumn<Packet, String> toColumn = new TableColumn<>("%toLabel");
    toColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("toLabel"));

```

```

        toColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("deviceTo"));

        TableColumn<Packet, String> channelColumn = new
TableColumn<>("%channelLabel");
        channelColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("channelLabel"));
        channelColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("channel"));

        TableColumn<Packet, String> dataRateColumn = new
TableColumn<>("%dataRateLabel");
        dataRateColumn.setText(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("dataRateLabel"));
        dataRateColumn.setCellValueFactory(new PropertyValueFactory<Packet,
String>("dataRate"));

        tableView.getColumns().setAll(timeColumn, typeColumn, fromColumn,
toColumn, channelColumn, dataRateColumn);

        tableView.setItems(packetList);
    }

    public static void loadPieChart() {
        PieChart.Data slice1 = new PieChart.Data("%standbyLabel", 0);
        slice1.setName(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("standbyLabel"));
        PieChart.Data slice2 = new PieChart.Data("%advertisingLabel", 0);
        slice2.setName(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("advertisingLabel"));
        PieChart.Data slice3 = new PieChart.Data("%connectedLabel", 0);
        slice3.setName(ResourceBundle.getBundle("view.lang",
Locale.getDefault()).getString("connectedLabel"));

        slice1.pieValueProperty().bind(device.getStandbyTime());
        slice2.pieValueProperty().bind(device.getAdvertisingTime());
        slice3.pieValueProperty().bind(device.getConnectedeTime());

        ObservableList<PieChart.Data> data =
FXCollections.observableArrayList(slice1, slice2, slice3);
        pieChart.setData(data);

        for (final PieChart.Data d : pieChart.getData()) {
            d.getNode().addEventHandler(MouseEvent.MOUSE_PRESSED, new
EventHandler<MouseEvent>() {
                @SuppressWarnings("unlikely-arg-type")
                @Override
                public void handle(MouseEvent e) {
                    caption.setVisible(false);
                    caption.setTranslateX(e.getSceneX());
                    caption.setTranslateY(e.getSceneY() - 100);

                    if (!Locale.getDefault().equals("bg"))
                        caption.setText(
                            d.getName() + " state
duration: " + String.valueOf(d.getPieValue()) + " seconds");
                    else
                        caption.setText(

```

```

d.getName() + ": " + String.valueOf(d.getPieValue()) + " " +
    "Продължителност в " +
    "секунди");
    caption.setVisible(true);
    }
    });
    }
    }
    }

```