Tampereen ammattikorkeakoulu

Bluetooth Low Energy -laitteiden ja Android-mobiililaitteiden välinen kommunikaatio

Terhi Salonen

OPINNÄYTETYÖ
Marraskuu 2022

Tietojenkäsittelyn tradenomi
Ohjelmistotuotanto

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn tradenomi
Ohjelmistotuotanto

SALONEN, TERHI:
Bluetooth Low Energy -laitteiden ja Android-mobiililaitteiden välinen kommunikaatio

Opinnäytetyö 92 sivua, joista liitteitä 15 sivua
Marraskuu 2022

Opinnäytetyön taustalla oli dokumentaation tarve, joka liittyy Bluetooth Low Energy -laitteiden ja Android-sovelluksen väliseen kommunikaatioon. Toimeksiantajayrityksellä sovelluskehittäjänä toimiva opinnäytetyön tekijä otti vastuulleen projektin, jossa korjattiin olemassa olevaa koodia ja kehitettiin uusia ominaisuuksia Android-laitteille kirjoitettuun Software Development Kitiin. SDK:n tehtävä oli hallinnoida Android-laitteen ja Bluetooth Low Energy -laitteiden välistä yhteyttä. Projektin dokumentaatio oli suunnattu lähinnä projektin asiakkaalle ja SDK:n käyttäjälle, minkä takia opinnäytetyön tekijä päätti ehdottaa toimeksiantajayritykselle, että hän tuottaisi opinnäytetyönään dokumentaation tukemaan vastaavan kaltaisen SDK:n kehittämistä.

Opinnäytetyön tarkoituksena oli laatia toimeksiantaja yritykselle käsikirjankaltainen ohjeistus siitä, miten Bluetooth ja Bluetooth Low Energy toimivat sekä, miten Bluetooth Low Energy -laite kommunikoi Android-laitteen kanssa. Opinnäytetyön tavoitteena oli syventää toimeksiantajayrityksen ohjelmistokehittäjien osaamista liittyen Bluetoothiin, Bluetooth Low Energyyn ja Bluetooth Low Energy -laitteiden hallinnointiin Android-mobiililaitteilla. Toimeksiantajana työlle toimi Haltu Oy. Opinnäytetyön tuloksena on kattava raportti työn aihealueesta sekä siitä jalostettu, kevyempi dokumentaatio (liite 1), joka lisätään toimeksiantajan käsikirjaan kaikkien yrityksen sovelluskehittäjien ulottuville.

Asiasanat: bluetooth, bluetooth low energy, android, sdk

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Bachelor of Business Information Systems
Software Production

SALONEN, TERHI:
The communication between Bluetooth Low Energy devices and Android mobile devices

Bachelor's thesis 92 pages, appendices 15 pages
November 2022

This thesis was based on a need of documentation regarding how communication between Bluetooth Low Energy devices and Android application is handled. The client of this thesis had a project, which was about debugging and developing new features for Software Development Kit used for handling communication between Android mobile devices and Bluetooth Low Energy devices. The documentation regarding the project was directed at the client who had ordered the SDK instead of the developers who developed it, which made the project unnecessary challenging to work with.

The aim of the thesis was to create handbook like instructions on how Bluetooth and Bluetooth Low Energy work and, how Bluetooth Low Energy devices communicate with an Android application. The thesis client was Haltu Oy. The goal was to deepen the knowledge regarding the subject of the thesis among the developers within the client company. As result, there is now a somewhat comprehensive report of the subject, but also more light weight documentation (appendices 1) for the purposes of the client’s handbook.

Key words: bluetooth, bluetooth low energy, android, sdk

SISÄLLYS

LYHENTEET JA TERMIT

ad hoc	Langattomien lähiverkkojen yhteystapa, joka ei vaadi tukiaseman käyttöä
AMP	Alternate MAC / PHY; ominaisuus, joka mahdollistaa Bluetooth 3.0 -laitteille Wi-Fi-yhteyden hyödyntämisen datasiirrossa
Android	Mobiililaitteiden käyttöjärjestelmä
API	Application Programming Interface; ohjelmointirajapinta, joka voi toimia joko ohjelmien välillä tai ohjelmien ja laitteiden välillä
Bluetooth, BT	Lyhyen kantaman langattoman yhteyden teknologia
Bluetooth LE, BLE	Bluetooth Low Energy
BR	Basic Rate; lähtökohtainen datansiirtonopeuden moodi
Characteristics	BLE GATT-protokollan mukainen datan tallennusyksikkö
dBm	Desibelimilliwatti; sähkötehon yksikkö desibeleinä
EDR	Enhanced Data Rate; datansiirtonopeutta parantava moodi
FHSS	Frequency-hopping Spread Spectrum; taajuushyp-pelyteknologia
GFSK	Gaussian Frequency Shift Keying; modulaatioskeema, jossa binäärinen 1 ja 0 sisällytetään kantoaallon pituuteen poikkeuttamalla kantoaallon perustaajuutta
GHz	Gigahertsi; taajuusyksikkö
HS (AMP)	High Speed; moodi, joka AMPia hyödyntämällä mahdollistaa suuremman datansiirtonopeuden
iOS	Applen mobiilikäyttöjärjestelmä
IoT	Internet of Things; esineiden internet
ISM	Industrial, scientific and medical; teolliset, tieteelliset ja lääketieteelliset luvasta vapaat radiolähettimet
kbps	Kilobittiä sekunnissa; siirtonopeuden määre

MAC	Medium Access Control Layer; Bluetoothin protokollapinon toiseksi alin kerros, joka tunnetaan myös nimellä Link Layer
Mbps	Megabittiä sekunnissa; siirtonopeuden määre
OSI	Open Systems Interconnection; tiedonsiirtoprotokollapino, jolla on seitsemän tasoa
p/4-DQPSK	Differential Quadrature Phase-Shift Keying; datansiirtoon liittyvä modulaatioskeema
PHY	Physical Layer; Bluetooth Low Energy:n protokollapinon alimmainen protokolla, joka sisältää analogiset kommunikaatiopiirit, joiden avulla BLE-laitteet viestivät
RF	Radio Frequency; radiotaajuus
SDK	Software Development Kit; ohjelmistokehityspaketti, eli ohjelmointiympäristöön asennettava, paketoitu kokoelma ohjelmointityökaluja
Services	BLE GATT-protokollan mukainen datan organisointiin liittyvä yksikkö
SIG	Bluetooth Special Interest Group; teknologia-alan ryhmittymä, joka alkoi kehittää ja kehittää yhä Bluetooth-teknologiaa
TCP/IP	Transmission Control Protocol / Internet Protocol; tiedonsiirtoprotokollapino, jolla on neljä tasoa
8DPSK	Differential 8-Phase Shift Keying; datansiirtoon liittyvä modulaatioskeema

1 JOHDANTO

Tässä opinnäytetyössä tarkastellaan, miten Bluetooth ja Bluetooth Low Energy toimivat ja miten Android-laite voi kommunikoida Bluetooth Low Energy -laitteiden kanssa. Lisäksi opinnäytetyössä selvitetään, miten Android-laitteelle voidaan luoda näiden laitteiden väliseen kommunikointiin tarvittavat ominaisuudet ja toiminnallisuudet sisällään pitävä Android-kirjasto, jota voidaan Software Development Kitin tavoin hyödyntää projektista toiseen. Opinnäytetyön toimeksiantajana toimi Haltu Oy.

Opinnäytetyön tarkoituksena on laatia toimeksiantajayritykselle käsikirjankaltainen dokumentaatio, siitä miten Bluetooth ja Bluetooth Low Energy toimivat ja miten Android-laite saadaan kommunikoimaan Bluetooth Low Energy-laitteiden kanssa. Opinnäytetyön tavoitteena on syventää toimeksiantajayrityksen sovelluskehittäjien osaamista opinnäytetyön aiheeseen liittyen.

Opinnäytetyön aiheen taustalla on opinnäytetyöntekijän kokemus työskentelystä Androidille kirjoitetun Bluetooth Low Energy -laitteiden kanssa kommunikoivan SDK:n parissa. Projektin dokumentaatio kattoi lähinnä SDK:n käytön asiakkaan näkökulmasta. Näin ollen projektin budjettia kului suotta sellaisen ohjelmoinnin kannalta tarpeellisen tiedon etsimiseen, mikä olisi voinut olla dokumentoituna joko projektin tiedostoissa tai projektin muiden dokumenttien joukossa.

Opinnäytetyön aihetta käsitellään teorian ja esimerkkikoodin avulla. Esimerkkikoodin kielenä on Kotlin, joka on natiivi ohjelmointikieli Android-käyttöjärjestelmälle. Työssä käydään ensin läpi, mitä ovat Bluetooth ja Bluetooth Low Energy ja miten ne toimivat. Tämän jälkeen selvitetään, miten Androidin Bluetooth Low Energy -sovellusrajapintoja voidaan käyttää Android- ja Bluetooth Low Energy -laitteiden väliseen kommunikaatioon. Lopuksi käydään vielä läpi, miten Androidille voi rakentaa kirjastomoduulin, jota voi käyttää SDK:n tavoin osana muita sovelluksia.

2 BLUETOOTH

Bluetooth on lyhyen kantaman langattoman yhteyden teknologia. Se hyödyntää yhteyksissään samoja radiotaajuuksia kuin esimerkiksi Wi-Fi-reititin, operoiden luvasta vapaiden radiolähettimien ISM-kaistalla (Industrial, scientific and medical), joka on keskitetty 2.4 gigahertsiin. (Scientific American 2007.) Bluetoothin ja Wi-Fin yhteneväisyydet jäävät kuitenkin muutoin vähäisiksi, sillä niiden käyttötarkoitukset eroavat merkittävästi toisistaan (Woodford 2021).

Bluetoothin toiminta nojaa FHSS-teknologiaan (frequency-hopping spread spectrum). Tämän teknologian avulla Bluetooth jakaa taajuuskaistan pienempiin kanaviin ja hyppii tiheästi, pseudosatunnaisella muuttuvalla kaavalla, kanavasta toiseen datasignaaleja siirtäessään. Sekä lähettävä että vastaanottava laite tuntevat muuttuvan kaavan, ja kanava vaihtuu aina yhden datasignaalin siirron jälkeen. (Hanna & Burke 2021.)

Perinteisesti Bluetoothin kantama-alueen ylärajana on mainittu 10 metriä (Woodford 2021). Bluetooth-teknologia on kuitenkin kehittynyt monin myös tällä saralla. Woolley (2021, 8) kertoo, että epävirallisessa testauksessaan hän sai vastaanotettua Bluetooth Low Energy -mikrokontrollerin lähettämiä ilmoituksia älypuhelimellaan yli 350 metrin kantaman päästä alueella, joka oli alle optimaalisen, pitäen sisällään puita, ihmisiä ja muuta radioliikennettä.

2.1 Bluetoothin historia ja kehitys

Bluetooth sai alkunsa 1990-luvulla Ericssonilla, kun teknologia-alan yritykset yrittivät kukin tahoillaan ratkaista lyhyen kantaman laiteyhteyksien haasteita, eli lähinnä päästä eroon johdoista, kaapeleista ja työläistä asennuksista, joita lisälaitteet ja -varusteet yhä edellyttivät. Aluksi Ericsson alkoi kehittää Bluetoothia ja alan muut suuret yritykset alkoivat kukin kehittää sille omaa vastinettaan. 90-luvun puolivälissä nämä yritykset kuitenkin ymmärsivät, että yhden teknologian standardointi hyödyttäisi kaikkia enemmän kuin se, että jokainen koettaisi kehittää omaa vaihtoehtoista tapaansa ratkoa samoja ongelmia. Vuonna 1998 perustettiin Bluetooth Special Interest Group (SIG), johon kuului

perustamishetkellä viisi yritystä: Ericsson, Intel, Nokia, IBM ja Toshiba. (Pothitos 2017.) SIG on kasvattanut jäsenmääräänsä vuosien varrella merkittävästi, jo ensimmäisen vuoden aikana siihen oli liittynyt yli 4000 jäsentä. Nykyään SIGillä on yli 33 000 jäsentä. (Marcel 2018.)

Bluetooth sai nimensä samalla kertaa, kun SIG perustettiin (Pothitos 2017). Se nimettiin 10. vuosisadan tanskalaisen kuninkaan Harald ”Bluetooth” Gormssonin mukaan. Kuningas Harald yhdisti aikanaan Tanskan ja Norjan, ja uuden teknologian nähtiin yhtenäistävän standardoinnin avulla tulevaisuudessa elektronisia ratkaisuja, laitteita ja varusteita. Bluetoothin logossa yhdistyvät riimut ᚼ ja ᛒ, jotka ovat kuningas Haraldin nimikirjaimet. (Peter 2021.)

2.1.1 Bluetooth-versiointi ja versioiden vertailu

Bluetoothin eri versioita tarkastellessa vertailukohteina käytetään yleensä kolmea eri tekijää: kantamaa, datansiirtonopeutta ja virrankulutusta. Nämä tekijät määräytyvät kunkin version käyttämän modulaatioskeeman ja datapaketin mukaan. Alun alkaen Bluetoothin käyttämä modulaatioskeema oli GFSK, Gaussian Frequency Shift Keying, jossa kantoaallon perustaajuutta vaihtelemalla saavutettiin 1 Mbps datasiirtonopeus ja kantamaksi 10 metriä. (Nguyen 2018.)

Bluetooth jaetaan versioidensa perusteella Bluetooth Classic -laitteisiin ja Bluetooth Low Energy -laitteisiin (BLE). Bluetooth-versiot 1.0–3.0 kuuluvat Classic -kantaan ja kaikki tähän mennessä julkaistut versiot 4.0:sta eteenpäin ovat BLE-kantaa. Bluetooth Classic ja BLE eivät ole yhteensopivia, vaan Classic-laitteet voivat keskustella vain Classic-laitteiden kanssa ja BLE-laitteet vain toisten BLE-laitteiden kanssa. Poikkeuksena tähän sääntöön ovat niin sanotut Dual Mode -laitteet, kuten monet älypuhelimet, jotka tukevat kumpaakin laitekantaa. (Get Connected 2021.)

2.1.2 Bluetooth Classic

Triggs ja Wankhede (2022) kirjoittavat, että Bluetooth 1.0 julkistettiin vuonna 1999 ja samana vuonna esiteltiin ensimmäinen Bluetooth-laite, langaton kuulokesetti mobiilipuhelimille. Ensimmäinen Bluetooth-puhelin oli Ericssonin T36, joka esiteltiin 2000. Tämä malli ei kuitenkaan koskaan päätynyt kauppoihin. Ensimmäinen myyntiin tullut Bluetooth-puhelin tuli markkinoille 2001. Se oli Ericssonin T39. (Triggs & Wankhede 2022.)

Bluetooth 2.0 näki päivänvalon 2004 tuoden mukanaan ominaisuuden EDR (Enhanced Data Rate), joka paransi Bluetoothin kantamaa ja kaistaleveyttä (Triggs & Wankhede 2022). Tähän versioon Bluetooth vaihtoi modulaatioskeeman GFSK:sta tuoreempiin skeemoihin p/4-DQPSK (Differential Quadrature Phase-Shift Keying) ja 8DPSK (Differential 8-Phase Shift Keying), jotka kantoaallon taajuuden vaihtelun sijasta hyödyntävät aallon muodonmuutosta informaation kuljetuksessa. Tällä tavoin Bluetooth 2.0 saavuttaa datansiirtonopeudeksi 2 Mbps, ja jopa 3 Mbps. (Nguyen 2018.)

Vuosi 2009 merkitsi Bluetooth 3.0 ilmestymistä. Tämä versio esitteli moodin HS (AMP), eli High Speed (Alternate MAC / PHY), joka paransi datansiirtonopeuksia entisestään hyödyntämällä Wi-Fi-yhteyttä datansiirrossa. ”Bluetooth 3.0 + HS” -sertifioidut laitteet kykenevät siirtymään Wi-Fi-yhteyteen Bluetooth-linkin kautta suoritetun kättelyn jälkeen. Tällä tavoin nämä laitteet saavuttavat merkittävästi suuremman datasiirtonopeuden kuin aiemmat Bluetooth-versiot. ”Bluetooth 3.0 + HS” -sertifioidut laitteet saavuttavat peräti 24 Mbps datasiirtonopeuden. Huomioitavaa on kuitenkin, että Bluetooth esitteli HS-sertifioidun Bluetoothin rinnalla myös rinnakkaisen version Bluetooth 3.0, joka ei kykene hyödyntämään tiedonsiirrossa Wi-Fi-yhteyttä vaan on rajoittunut ns. tavallisen Bluetoothin tiedonsiirtomenetelmään. (GSMArena Team 2010.)

Bluetooth kehittyi etenkin datansiirtonopeuden saralla versioiden 1.0–3.0 aikana. Näiden versioiden, jotka tunnetaan myös nimellä Bluetooth Classic, käytettävyyttä heikentää kuitenkin etenkin suuri virrankulutus. Nopeasti tyhjenevistä akuista johtuen Bluetooth Classic ei sovellu IoT-käyttöön. (Nguyen 2018.)

2.1.3 Bluetooth Low Energy

Seuraava Bluetooth-versio poikkesi aiemmista siten, ettei se ollut SIGin kehittämä. Nokian matalan virrankulutuksen lähiyhteys -projekti, Wibree, sulautettiin osaksi Bluetoothia 2010. Näin syntyi versio, joka tunnetaan nimillä Bluetooth 4.0 ja Bluetooth Low Energy (BLE). (Peter 2021.)

Bluetooth 4.0 -spesifikaatio pitää itseasiassa sisällään sekä Classic- että Low Energy -teknologian, eli version moodit ovat BR (Basic Rate), EDR (Enhaced Data Rate), AMP (Alternate MAC/PHY), ja LE (Low Energy). Näin ollen SIG erotteli LE-teknologiaa käyttävät laitteet lisänimikkeillä ”Bluetooth Smart” ja ”Bluetooth Smart Ready”. Ensimmäinen näistä merkitsi laitteita, jotka käyttävät yksinomaan LE-teknologiaa ja jälkimmäinen merkitsi vuorostaan dual-laitteita, jotka tukivat sekä Classic- että LE-variaatioita. Tämä jaottelu sekoitti niin kuluttajat kuin kehittäjätkin, joten SIG luopui siitä lopulta ja niputti kaikki tämän version tyypit Bluetooth 4.0 -kattotermin alle. Kuopattuja termejä ”Bluetooth Smart” ja ”Bluetooth Smart Ready” vilisee kuitenkin edelleen laajasti levinneissä materiaaleissa. (Get Connected Blog 2021.)

BLE:n myötä esiteltiin uusi topologia, Broadcast. Tähän liittyen ISM-kaistalle tehtiin uusi jako, jonka myötä 79 kanavan sijasta BLE-laitteet käyttävät 40 kanavaa, joista kolme on Advertisement-kanavia. Advertisement on BLE:n esittelemä ominaisuus, jonka avulla BLE-laitteet lähettävät (broadcast) ”mainosdataa”, eli kertovat vastaanottavalle laitteelle läsnäolostaan (Tawil 2018). Advertisement-ominaisuutta käsitellään tarkemmin luvussa 3.

BLE pudotti siirtonopeuden takaisin lähelle 1 Mbps nopeutta, mutta vähensi merkittävästi virrankulutusta ja tyypillistä latenssia sekä kasvatti kantaman 100 metriin. Lisäksi BLE esitteli uuden topologian, point-to-multipoint, jonka ansiosta sitä tukevat Bluetooth-kuulokkeet tai -kaiuttimen saa yhdistettyä useampaan äänilähteeseen samanaikaisesti, siis esimerkiksi puhelimeen ja tietokoneeseen. (Peter 2021.)

Vuonna 2016 julkaistu Bluetooth 5.0 lukeutuu niin ikään BLE-versioksi. Se parantaa BLE-standardeja kasvattamalla datansiirtonopeutta ja kantamaa

kuitenkin säilyttäen edelleen matalan virrankulutuksen. Poiketen aiemmista versioista, Bluetooth 5.0 tarjoaa erilaisiin siirtotarpeisiin neljä eri datansiirtonopeutta: 2 Mbps, 1 Mbps, 500 kbps ja 125 kbps. Koska pidempi kantama edellyttää suppeampaa datansiirtonopeutta, versioon lisättiin tämän mahdollistava 125 kbps siirtonopeus. Esimerkiksi Bluetooth-sensorin ei välttämättä tarvitse lähettää suurta datamäärää kerralla, mutta se hyötyy pitkästä kantamasta, joka voi Bluetooth 5.0:lla olla peräti 240 metriä. Toisaalta toisenlaiset laitteet saattavat hyötyä suuresta datasiirtonopeudesta pikemmin kuin pitkästä kantamasta, jolloin 2 Mbps tulee tarpeeseen. (Nguyen 2018.)

Bluetooth 5.0 myötä BLE sai jälleen uuden topologian, Mesh-yhteyden avulla BLE-laitteilla voidaan rakentaa laajan skaalan verkkoja. Mesh-verkkojen kautta voidaan ohjata, monitoroida ja automatisoida järjestelmiä, joissa on satoja tai jopa tuhansia laitteita, jotka keskustelevat keskenään. Mesh-yhteyden ominaisuuksia on kuvattu kuviossa 1. (Bluetooth, 2022.)

KUVIO 1. Mesh-yhteyden ominaisuudet (Bluetooth, 2022, muokattu).

Bluetooth 5.0 mahdollistaa edellä mainittujen asioiden lisäksi kaksi BLE Advertising -moodia, jotka ovat Legacy Advertising (alkuperäinen moodi) ja Extended Advertising. Jälkimmäinen kasvattaa Advertising-datapaketin koon 31 tavusta 254 tavuun, käyttää Advertising-kanavien lisäksi myös yleiseen datasiirtoon tarkoitettuja 37 kanavaa Advertising-paketin lähettämiseen (broadcast) sekä synkronoi Advertising-paketin siirron lähettävän ja vastaanottavan laitteen välillä. (Bluetooth, 2022.)

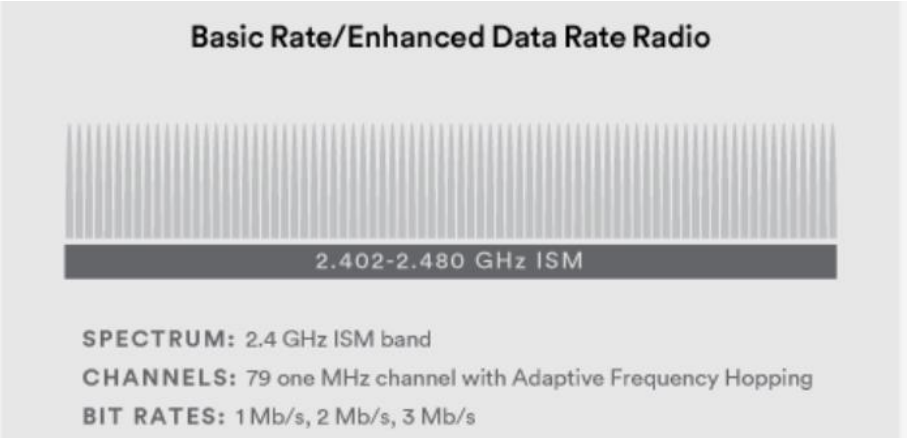
2.2 Bluetoothin toiminta

Bluetoothin toiminta rakentuu monista osista. Siihen vaikuttaa fyysinen radiolähetin, protokollat, jotka määrittelevät ja ohjaavat kuinka Bluetooth-laite toimii, profiilit, jotka kertovat mihin laitetta käytetään, topologia, joka määrittelee millaisissa verkoissa laite voi olla mukana ja missä roolissa se voi esiintyä. Näiden lisäksi Bluetoothin toiminnassa huomioidaan tietenkin myös tietoturva.

2.2.1 Radio

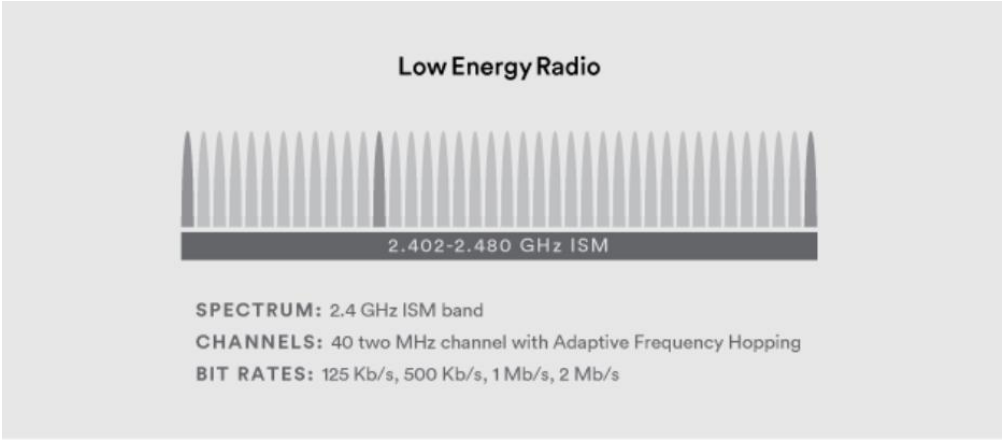
Bluetooth-radio operoi 2.4 GHz:n ISM-kaistalla, joka on jaettu joko 79 kanavaan, Bluetooth Classic (kuva 2), tai 40 kanavaan, BLE (kuva 3). Bluetoothin toiminta perustuu FHSS-teknologiaan, eli pseudosatunnaiseen taajuushyppelyyn, joka suojaa kanavahäirinnältä (interference), siirrettävien datasignaalien kaappaukselta (interception) ja dataliikenteen jumittamiselta ja häirinnältä (jamming). (Hanna & Burke 2021.)

Bluetooth Classicin siirtonopeus varioi 1–3 Mbps:n välillä versionumerosta riippuen (kuva 2). Siitä on tullut standardi radioprotokolla langattomien äänentoistolaitteiden keskuudessa. (Bluetooth, 2022.)



KUVA 2. Bluetooth Classic -laitteiden radioyhteyden spesifikaatiot (Bluetooth, 2022, muokattu).

Bluetooth Low Energyn käyttämästä 40 kanavasta kolme on Advertising-kanavia ja 37 kanavaa käytetään yleiseen datansiirtoon (kuva 3). Sen siirtonopeudet varioivat 125 kbps:n ja 2 Mbps:n välillä. (Bluetooth, 2022.)



KUVA 3. Bluetooth Low Energy -laitteiden radioyhteyden spesifikaatiot (Bluetooth, 2022, muokattu).

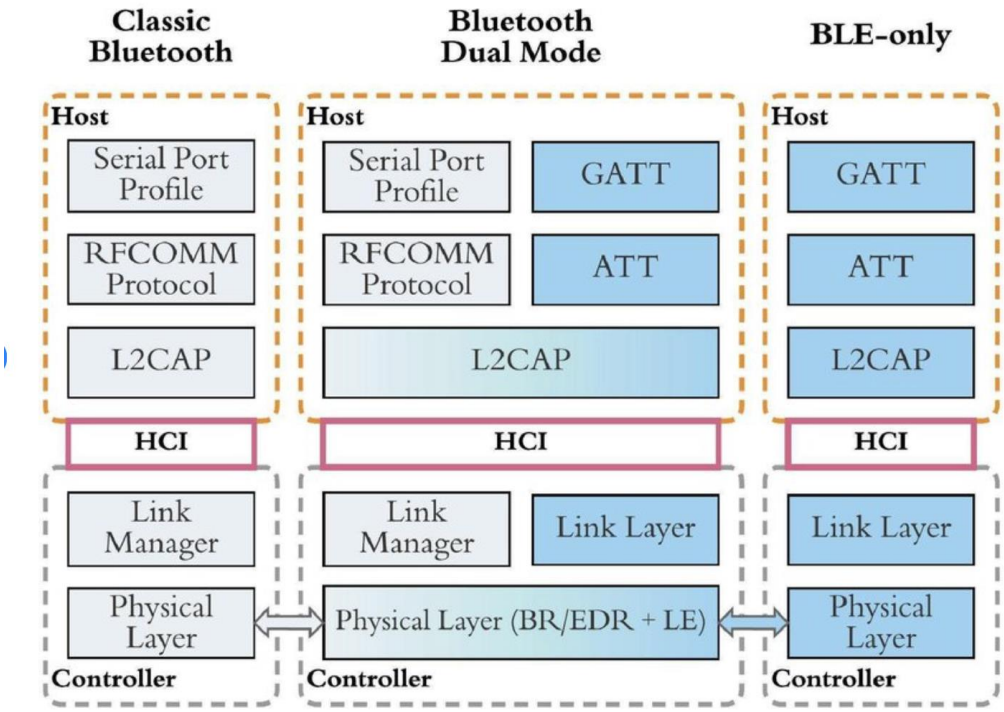
Bluetooth-lähettimen teho ja kantama ilmaistaan luokkajärjestelmällä. Suurempi teho tarkoittaa myös suurempaa kantamaa. Toisaalta mitä pienempi Bluetooth-luokka on kyseessä sitä suurempi teho, ja siten myös kantama, laitteella on. Bluetooth-luokat jakautuvat seuraavasti:

- Luokka 1: 100 megawattia (20 desibelimilliwattia), 100 metriä
- Luokka 2: 2,5 mW (4 dBm), 10 m
- Luokka 3: 1 mW (0 dBm), 1 m. (Andy G. 2022.)

2.2.2 Protokollat

Bluetooth-arkkitehtuuri ei noudata standardoitua OSI- tai TCP/IP-mallia, vaan sillä on oma itsenäinen malli protokollapinolleen. Bluetooth poikkeaa perinteisistä verkkoarkkitehtuureista myös siten, ettei sitä tukevien laitteiden tarvitse toteuttaa kaikkia pinon protokollia, vaan Bluetoothia hyödyntävä sovellus määrittelee sen, mitä niistä se käyttää. (Moumita 2020.)

Bluetooth Classicin protokollapino eroaa jonkin verran BLE:n vastaavasta. Edellä esitetty arkkitehtoninen periaate pätee kuitenkin molemmissa tapauksissa. Molempien osalta pätee myös se, että protokollapinoille tyypillisesti alin kerros on fyysinen kerros, siis raudan (hardware) protokollia tai sen kanssa kommunikoivia protokollia. Mitä ylemmäs pinossa noustaan, sitä lähemmäs käyttäjärajapintaa siirrytään. Bluetooth Dual Mode -laitteet tukevat kumpaakin protokollapinoa, kuten on esitetty kuviossa 4.

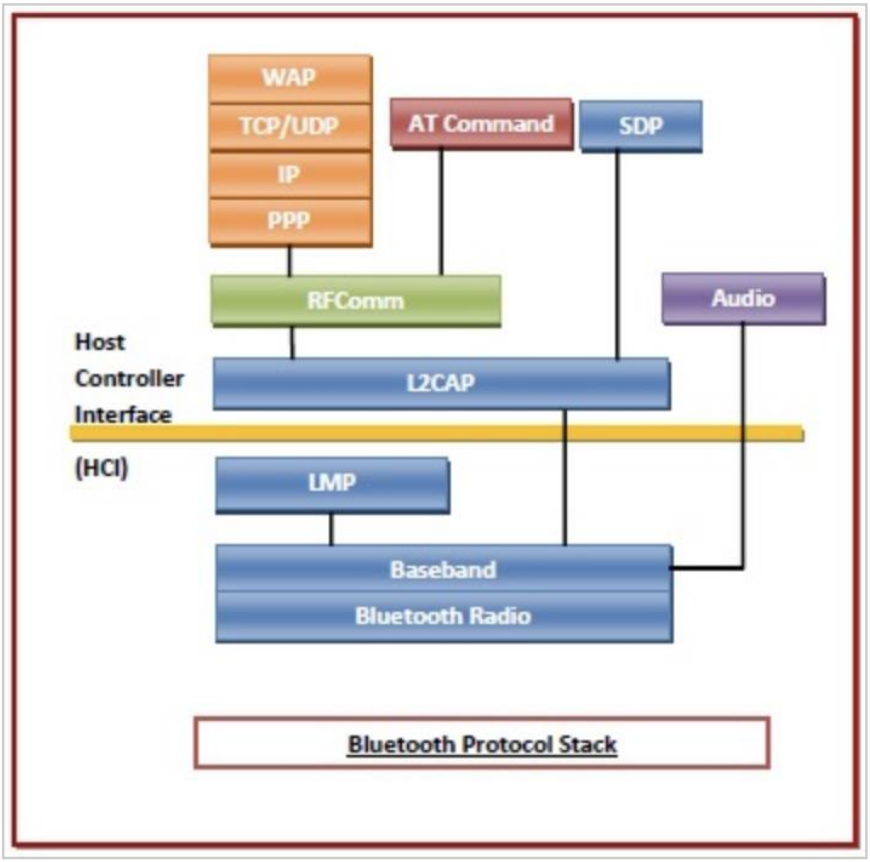


KUVIO 4. Erilaiset Bluetooth protokollapinot (Yang, Poellabauer, Mitra & Neubecker 2019, 3).

Bluetooth Classic

Bluetooth Classic -pino (kuvio 5) voidaan jakaa kolmeen osaan. Nämä osat ovat Physical Layer, Data Link Layer ja Middleware Layer (Moumita 2020). Tässä opinnäytetyössä Bluetooth Classic -protokollapino käydään läpi tämän kolmijakoisen jaottelun mukaan, mutta protokollapinoa voidaan toisaalta tarkastella myös kahden osan, controller ja host, kautta. Kaksijakoisessa jaottelussa controller-osa kuvastaa pinon alimpia protokollia, Radio, Baseband ja Link Manager. Host-osaan lukeutuvat vuorostaan kaikki ylemmän tason protokollat (Tawil 2017).

Kolmijakoisessa jaottelussa Physical Layer on pinon pohjalla, Data Link Layer keskellä ja Middleware Layer muodostaa pinon uloimman kerroksen. Osa protokollista on näiden osien rajapinnassa siten, että ne vaikuttavat sekä rajapinnan ylä- että alapuolella olevassa osassa. (Moumita 2020.)



KUVIO 5. Bluetooth Classic -protokollapino (Moumita 2020).

Physical Layer pitää sisällään protokollat Bluetooth Radio ja Baseband, joista jälkimmäinen vaikuttaa myös seuraavassa osassa (Data Link Layer). Bluetooth Radio -protokolla vastaa radioaaltojen fyysisestä rakenteesta ja määrittelyistä. Se määrittelee muun muassa taajuuskaistat, FHSS-taajuushyppelyn toimintamallin ja käytettävät modulaatiot. Baseband-protokollan tehtävänä on vuorostaan vastata radioprotokollan palveluista, eli esimerkiksi ajoituksesta, osoitejärjestelmästä ja virransäästöalgoritmeista. (Moumita 2020.)

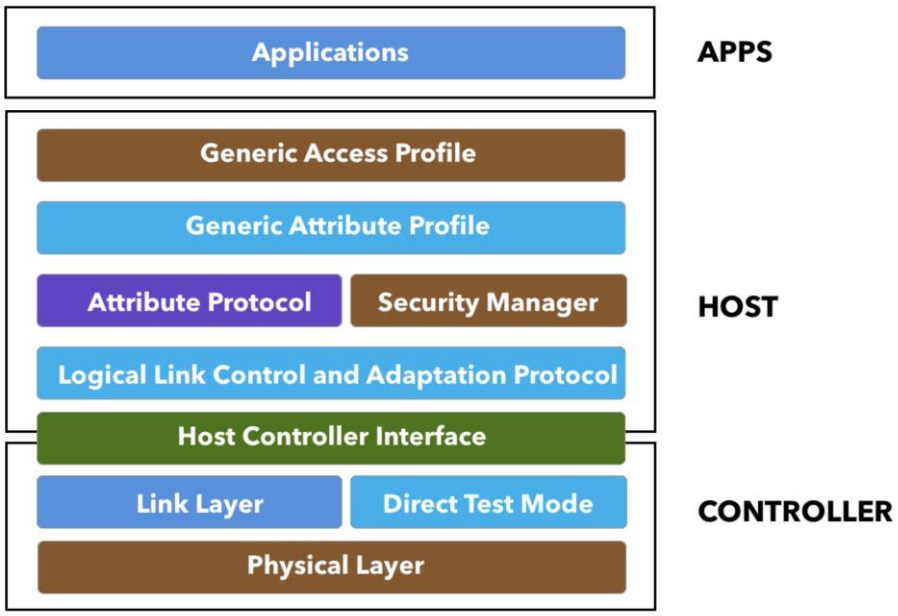
Data Link Layer sisältää protokollat Baseband, Link Manager Protocol (LMP) ja Logical Link Control and Adaptation Protocol (L2CAP). Host Controller Interface (HCI), alemman ja ylemmän tason välinen rajapinta, kulkee LMP:n ja L2CAPin välissä. LMP jää siis alemmalle tasolle (Controller) ja L2CAP sijaitsee vuorostaan ylemmällä tasolla (Host). Näin ollen LMP perustaa loogiset linkit (logical links) Bluetooth-laitteiden välille ja ylläpitää niitä mahdollistaen täten laitteiden välisen kommunikaation. LMP vastaa myös laiteautentikaatiosta, viestien salaamisesta sekä datapakettikoon neuvottelusta. L2CAP puolestaan sovittaa Baseband-protokollan ylemmän ja alemman tason yhteen. (Moumita 2020.)

Middleware Layer koostuu protokollista Radio Frequency Communication (RFComm), Adopted Protocols, Service Discover Protocol (SDP) ja AT Commands (AT). RFComm vastaa radiokomponentin emuloiduista sarjaporteista. Adopted Protocols pitää sisällään muualta adoptoituja protokollia. Näistä yleisimmin Bluetoothin käytössä ovat Point-to-Point Protocol (PPP), Internet Protocol (IP), User Datagram Protocol (UDP), Transmission Protocol (TCP) ja Wireless Application Protocol (WAP). SDP vastaa palveluihin liittyvistä kyselyistä, kuten esimerkiksi laiteinformaatiosta, jonka perusteella Bluetooth-laitteet voivat muodostaa yhteyden toisiinsa. (Moumita 2020.) Middlewareen kuuluva protokolla AT on aikanaan kehitetty puhelinmodeemien hallintaa varten. Bluetoothissa AT:ta voidaan käyttää esimerkiksi hallitsemaan ja muokkaamaan nykyisten ja tulevien käynnistyssyklien käytöstä ja asetuksia. (LM Technologies 2020.)

Bluetooth Low Energy

Bluetooth Low Energyn protokollapino voidaan jakaa karkeasti kolmeen osaan: Controller (pinon alakerta), Host (pinon yläkerta) ja Application (käyttäjärajapinta, sovellus). Controller- ja Host-osan välissä sijaitsee myös vapaavalintainen Host

Controller Interface (HCI). HCI mahdollistaa Host- ja Controller-osan välille standardimuotoisen kommunikaatiorajapinnan. Mikäli se ei ole käytössä, on sekä Host- että Controller-osa implementoitu samassa Bluetooth-sirussa. Jos HCI on käytössä, kuuluu Host-osa yhdelle Bluetooth-sirulle ja Controller-osa toiselle Bluetooth-sirulle, ja HCI vastaa näiden keskinäisestä kommunikaatiosta. (Hlapisi 2022.) Kuviossa 6 on esitetty BLE-protokollapino edellä kuvatun kolmijaon mukaan. Kuviossa on mukana myös vapaavalintainen HCI.



KUVIO 6. BLE-protokollapino (Afaneh 2018, 20).

Controller-osaan lukeutuvat:

- Physical layer (LE PHY)
- Link Layer (LL). (Hlapisi 2022.)

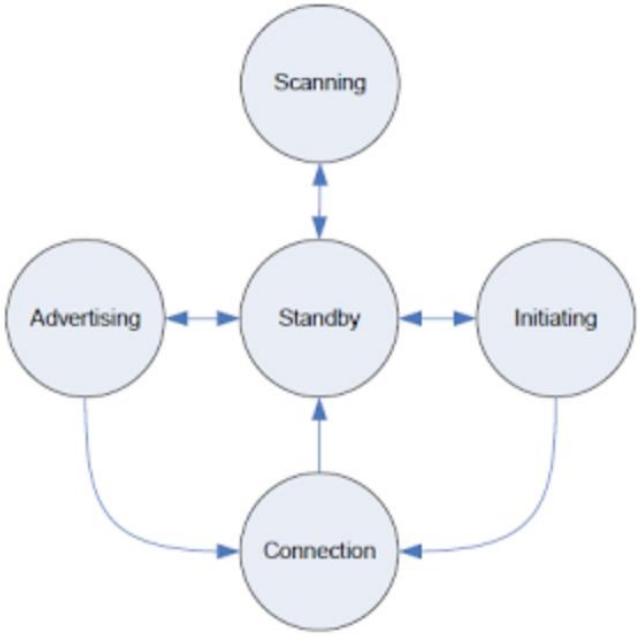
Controller-osasta löytyy fyysinen radio, joka vastaa radioaalloista, niiden luonnista, salauksesta, vastaanottamisesta ja salauksen purusta. Link Layer vastaa muun muassa BLE-radion tilanhallinnasta. (Hlapisi 2022.)

Link Layer -tason tarjoamat tilat ovat Standby, Advertising, Scanning (aktiivinen ja passiivinen), Initiating ja Connection (Master, Slave). Scanning-tilalla on siis kaksi vaihtoehtoista tilaa, aktiivinen ja passiivinen. Connection-tilalla on niin ikään kaksi vaihtoehtoista tilaa, Master (Central) ja Slave (Peripheral). LL määrittää

tilojen avulla BLE-laitteen roolin. Riippuen LL:n määrittämistä tiloista, BLE-laite voi olla:

- Advertiser: näitä ovat laitteet, jotka välittävät Advertising-datapaketteja Advertising-kanavia pitkin.
- Scanner: näitä ovat laitteet, jotka vastaanottavat Advertising-datapaketteja Advertising-kanavilta ilman aietta muodostaa yhteyttä lähettävän laitteen kanssa.
- Initiator: näitä ovat laitteet, jotka yrittävät aktiivisesti muodostaa yhteyden toisen BLE-laitteen kanssa ja täten koettavat kuuntelemalla löytää yhdistämisen hyväksyviä Advertising-datapaketteja.
- Central / Peripheral: yhteyden muodostamisen jälkeen, Advertiser-laite siirtyy Peripheral-tilaan ja Initiator-laite siirtyy Central-tilaan. (Su 2016.)

Kuviossa 7 on esitetty, miten LL hallinnoi BLE-laitteen eri tiloja. Kuviossa ei ole eritelty Scanning-tilan eikä Connection-tilan vaihtoehtoisia muotoja.



KUVIO 7. Link Layerin hallinnoimat tilat (Afaneh 2018, 23).

BLE-laitteiden kolme pääasiallista tilaa ovat Advertising, Scanning ja Connection. BLE-laitteet lähettävät Advertising-datapaketteja, jotta ne löytyvät Scanning-laitteilla, jotka voivat muodostaa yhteyden laitteiden välille ja siirtää siten

molemmat laitteet Connection-tilaan. Muut LL:n hallinnoimat tilat asettuvat näiden kolmen päätilan välimaastoihin. Standby-tilassa BLE-laite ei lähetä eikä vastaanota datapaketteja, tämä on BLE-laitteiden lähtökohtainen tila. Initiating-tila on vuorostaan tila Scanning- ja Connection-tilan välissä. Se on tila, joka astuu voimaan, kun Scanning-laite päättää muodostaa yhteyden Advertising-laitteen kanssa. (Afaneh 2018, 22–23.)

Host-osaan kuuluvat:

- Logical Link Control & Adatation (L2CAP)
- Security Manager (SM) ja Attribute Protocol (ATT)
- Generic Access Profile (GAP) ja Generic Attribute Profile (GATT). (Hlapisi 2022.)

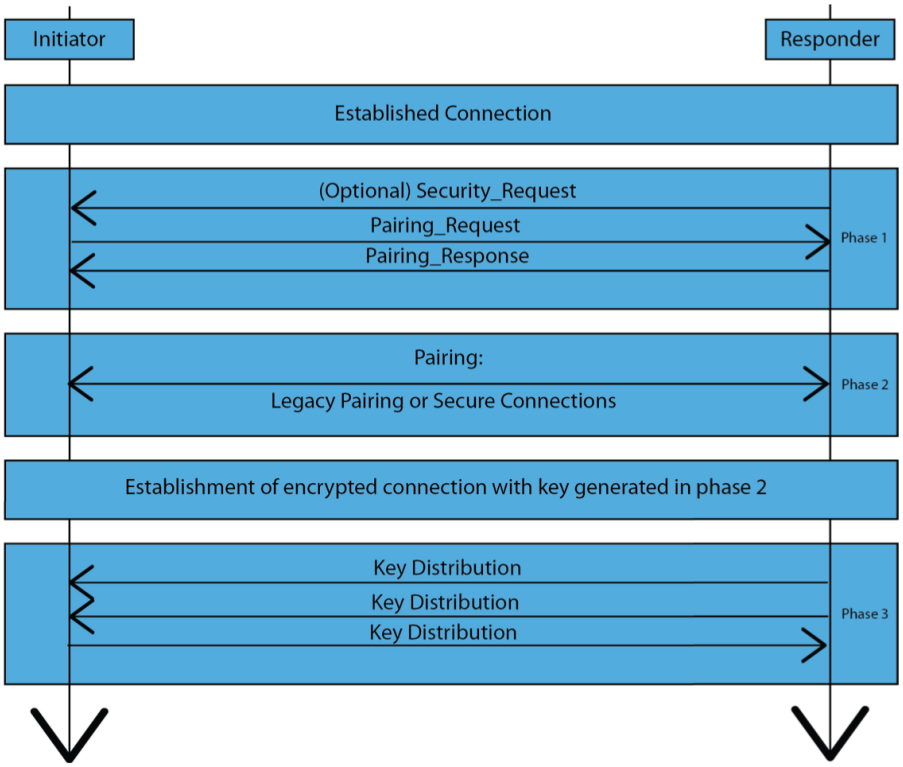
Host-osa on tyypillisesti ohjelmisto, joka muodostuu BLE-pinon ylimmistä kerroksista sekä BLE-profiileista. BLE-profiilit pitävät sisällään yksityiskohtaisen tiedon siitä, miten eri protokollien tulee keskustella ja työskennellä keskenään toteuttaakseen tietyn toimintamallin. (Hlapisi 2022.)

BLE-laitteen L2CAP-protokolla lainaa toimintamallinsa Bluetooth Classicin vastaavalta protokollalta. Sen pääasiallisesti käsittelemät ylemmän tason protokollat ovat Security Manager ja Attribute Protocol. (Afaneh 2018, 26.)

Security Manager (SM) määrittelee protokollat ja algoritmit, joiden perusteella yhteyttä muodostamassa olevat BLE-laitteet luovat ja vaihtavat avaimia. Se sisältää viisi turvallisuusominaisuutta, jotka ovat paritus (pairing), kiinteän yhteyden muodostus (bonding), todennus (authentication), salaus (encryption) ja viestien eheys (message integrity). (Afaneh 2018, 75.)

Paritus ja kiinteän yhteyden muodostus kulkevat käsikkäin, kuten kuviossa 8 on esitetty. Paritus muodostuu kuvion vaiheista yksi ja kaksi, kun taas vaihe kolme kattaa kiinteän yhteyden muodostuksen prosessit. Paritus on väliaikainen turvallisuustoimi, joka ei säily yhteyden katkettua. Paritus suoritetaan alusta loppuun joka kerta, kun kaksi BLE-laitetta muodostavat yhteyden, jonka haluavat salata. Kiinteä yhteys muodostetaan sellaisten laitteiden välille, joiden halutaan säilyttävän salatun yhteytensä yli yhteyden katkaisun ja uudelleen yhdistämisen.

(Afaneh 2018, 77.) BLE-laitteet voivat olla yhteydessä toisiinsa ilman kiinteää yhteyttä ja ilman paritusta, mutta tällöin niiden välillä siirtyvä data ei ole salattua. Opinnäytetyön taustalla olevassa projektissa laitteiden parituksesta teki päätöksen sovelluksen käyttäjä.



KUVIO 8. Turvallisuusprosessin vaiheet (Afaneh 2018, 77).

Vaiheessa kaksi on käytettäville metodeille kaksi vaihtoehtoa. Nämä vaihtoehdot ovat Legacy Pairing ja Secure Connections. Näistä ensimmäinen on tuettu kaikilla BLE-versioilla ja jälkimmäinen BLE-versiosta 4.2 alkaen. (Afaneh 2018, 77, 80–81.)

Vaihtoehtoiset paritusmetodit:

- **Legacy Pairing**: Legacy-yhteydet käyttävät kahta avainta, temporary key (TK) ja short term key (STK). TK:ta käytetään muiden vaihdettavien arvojen ohella STK:n luonnissa, yhteyttä muodostavien kahden laitteen välillä.
- **Secure Connections**: Tämä paritusmetodi ei edellytä avainten vaihtoa yhteyttä muodostavien BLE-laitteiden välillä. Sen sijaan se hyödyntää kummallakin laitteella ECDH-protokollaa (Elliptic-curve Diffie–Hellman) ja luo julkisen ja salaisen avaimen parin (public / private key pair). Tämän

jälkeen laitteet vaihtavat vain julkiset avaimet, joiden perusteella ne generoivat jaetun salaisen avaimen, long term key (LTK). (Afaneh 2018, 79.)

Vaiheessa yksi tapahtuva laitteiden välinen keskustelu määrittää sen, mitä paritusmetodia käytetään. Kuten edellä jo mainittiin, kaikki BLE-laitteet tukevat Legacy Pairing -metodia, mutta BLE 4.2 -laitteet, ja laitteet sitä suuremmalla versionumerolla, tukevat myös Secure Connections -metodia. Secure Connection -metodia käytetään ainoastaan siinä tapauksessa, että kumpikin yhteyttä muodostavista BLE-laitteista tukee sitä. Vaikka nämä erityyppiset paritusmetodit sisältävät samoin nimettyjä prosesseja, eroavat nämä prosessit ja niihin liittyvä datasiirto toisistaan. (Afaneh 2018, 80–81.)

Legacy Pairing -metodi käyttää avaimia TK ja STK. Legacy Pairing -metodi sisältää seuraavia prosesseja:

- **Just works**: Turvattomin kaikista Bluetoothin yhdistämisvaihtoehdoista. TK:n arvoksi asetetaan 0.
- **Out of Bounds (OOB)**: TK:n vaihto-operaatio suoritetaan muulla kuin BLE-teknologialla, yleisimmin käytetään near field communication (NFC) -teknologiaa. Tämä on turvallisin kaikista Legacy Pairing -metodin yhdistämismetodeista.
- **Passkey**: Tämä metodi edellyttää, että käyttäjä syöttää toisen yhteyttä muodostavan BLE-laitteen esittämän TK:n kuudesta luvusta muodostuvan numeroarvon toiseen yhteyttä muodostavaan BLE-laitteeseen. Tätä metodia voi käyttää siis vain silloin, jos ainakin toinen yhteyttä muodostavista laitteista omaa näytön ja näppäimistön, jolla laitteelle voi kirjoittaa. (Afaneh 2018, 80–81.)

Secure Connections -metodin alla olevat prosessit ovat seuraavat:

- **Just works**: BLE-laitteet vaihtavat julkiset avaimet muut generoidut arvot, joiden perusteella ne generoivat jaetun salaisen avaimen.
- **Out of Band (OOB)**: BLE-laitteet käyttävät arvojen vaihtamiseen muuta kuin BLE-teknologiaa. Jos käytetty teknologia on turvallisempaa kuin BLE-teknologia, muodostuu laitteiden välille entistä turvallisempi yhteys.

- **Passkey**: Tässä metodissa käytetään identtisiä kuusilukuisia numeroarvoja, jotka syötetään joko kumpaankin BLE-laitteeseen tai toinen yhteyttä muodostavista BLE-laitteista generoi sen ja käyttäjä syöttää saamansa arvon toiseen yhteyttä muodostavaan BLE-laitteeseen.
- **Numeric Comparison**: Tämä on turvallisin kaikista BLE-laitteiden yhteyden muodostamisen metodeista. Tämä metodi toimii kuten Secure Connection -metodin Just Works -metodi, jonka päälle on lisätty yksi turvallisuuskerros. Tämä kerros suojaa yhteyttä man-in-the-middle-hyökkäyksiä (MITM) vastaan. MITM-hyökkäyksissä laitteiden väliselle viestintäreitille tunkeutuu salaa kolmas osapuoli, joka esittää kummallekin laitteelle olevansa asianmukainen toinen laite. (Afareh 2018, 81.)

Kiinteän yhteyden muodostus on paritukseen linkittyvä, vaihtoehtoinen prosessi. Se edellyttää Secure Connection -metodin tukea ja käyttöä. Kiinteä yhteyden muodostus johtaa siihen, ettei yhteyden muodostaneiden BLE-laitteiden tarvitse jatkossa käydä läpi paritusprosessia luodakseen turvallisen kommunikaatioväylän. Kiinteän yhteyden muodostuksessa kumpikin BLE-laite tallettaa avainsarjan, jota voidaan jatkossa käyttää yhteyden muodostamiseen ja täten jättää väliin paritusprosessi. Nämä avaimet vaihdetaan laitteiden välillä sellaista salattua linkkiä pitkin, joka on salattu Secure Connection -metodin tuottamilla LTK-avaimilla. (Afareh 2018, 80.)

Todennusvaihe varmistaa, että yhteyttä muodostamassa olevat BLE-laitteet jakavat keskenään samat salaiset avaimet. Salaus on vuorostaan prosessi, jossa salataan laitteiden välillä lähettävä data käyttäen 128-bit AES Encryption -standardia, joka luo symmetrisen avaimen algoritmin. Tämä tarkoittaa sitä, että samaa avainta käytetään sekä salaukseen että salauksen purkuun molemmilla yhteydessä olevilla BLE-laitteilla. Viestien eheys on prosessi, jossa dataa lähettävä BLE-laite allekirjoittaa lähetettävän datan ja vastaanottava BLE-laite varmentaa vastaanottamansa datan allekirjoituksen. (Afaneh 2018, 75.)

Host-osaan sisältyy edellä esitettyjen protokollien lisäksi vielä Attribute Protocol (ATT), Generic Access Protocol (GAP) ja Generic Attribute Protocol (GATT). Ne muodostavat BLE-protokollapinon ylimmät kerrokset ja siten vaikuttavat suoraan siihen, miten BLE-laitteet kommunikoivat keskenään. ATT sallii BLE-laitteen

paljastavan tiettyjä sisältämänsä datan osia, eli attribuutteja, toiselle BLE-laitteelle. GATT on vuorostaan Service-kehys (Framework), joka määrittelee aliproseduurit, joilla ATT-tasoa käytetään. GAP kattaa BLE-profiilien ja sovelluksen välisen rajapinnan. Se vastaa laitteiden havaitsemiseen ja laitteiden välisen yhteyden muodostukseen liittyvistä toiminnoista (Services). (Texas Instruments 2016.) Näitä protokollia käsitellään tarkemmin luvussa 3.

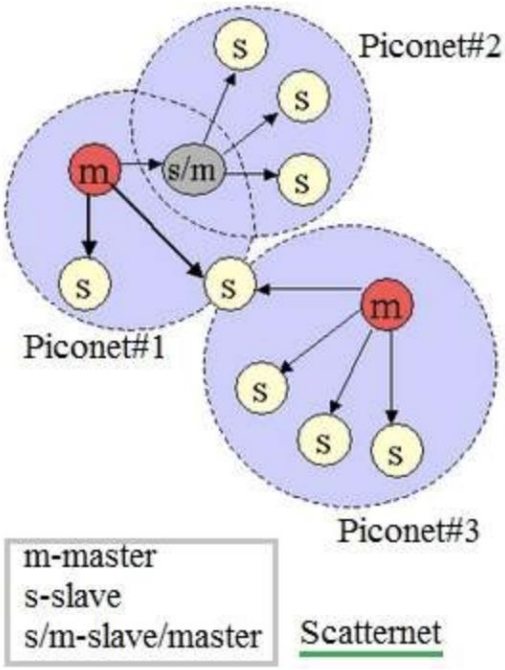
Application-osa toimii käyttäjän ja laitteen välisessä rajapinnassa. Se sisältää ohjelmiston, joka niputtaa koko protokollapinon muotoon, jossa se on käyttäjän käytettävissä. Se siis sisältää käyttöliittymän. (Hlapisi 2022.)

2.2.3 Topologia

Bluetooth toimii laitteiden välisellä Point-to-Point-yhteydellä, joka muodostaa Piconetin, langattoman lähiverkon (Bluetooth 2022). Toisiinsa yhteydessä olevat Bluetooth-laitteet muodostavat siis toisin sanoen ad hoc -verkon, jossa voi olla saman aikaisesti yksi Master-laite ja korkeimmillaan seitsemän Slave-laitetta (kuvio 9). Ad-hoc-verkot, ovat langattomia lähiverkkoja, jotka eivät käytä tukiasemaa.

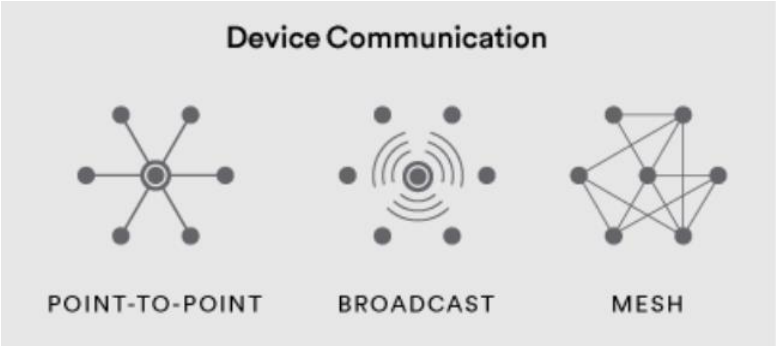
Piconetissa olevat laitteet, riippumatta siitä ovatko ne Master- vai Slave-laitteita, voivat kuulua samanaikaisesti muihinkin Piconeteihin joko samassa tai eri roolissa (Scientific American 2007). Mikäli Bluetooth-laite tai useampi kuuluu useampaan Piconetiin, ne muodostavat Piconet-verkoston, jota kutsutaan nimellä Scatternet (kuvio 10). (RF Wireless World 2012).

KUVIO 9. Piconet, tähtitopologia, Bluetooth-lähiverkon topologia (Bluetooth, 2022, muokattu).



KUVIO 10. Scatternet (RF Wireless World 2012).

Bluetooth Low Energy -teknologia tukee laajempaa valikoimaa erilaisia verkkotopologioita kuin Bluetooth Classic (kuvio 11). Monipuolisemman topologiavalikoimansa ansiosta BLE-laitteilla on mahdollista esimerkiksi rakentaa laajan skaalan laiteverkkoja Mesh-topologiaa hyödyntämällä. (Bluetooth 2022.)



KUVIO 11. Bluetooth Low Energy -lähiverkkojen topologiavaihtoehdot (Bluetooth, 2022, muokattu).

BLE-laitteet tukevat siis kahden laitteen muodostamaa kaksisuuntaista Point-to-Point-verkkoa (1:1), aivan kuten Bluetooth Classic -laitteetkin. Tämän lisäksi BLE-laitteet tukevat useamman BLE-laitteen muodostamia yhdensuuntaisia one-to-many-verkkoja (1:m). Tätä topologiaa kutsutaan nimellä Broadcast. Mesh-topologian verkot ovat vuorostaan kahdensuuntaisia many-to-many-verkkoja (m:m). (Lindevall 2022, 11–12.)

2.2.4 Bluetooth-profiilit

Bluetooth-protokollat määrittelevät kuinka laite toimiin, mutta Bluetooth-profiilit määrittelevät mihin laitetta käytetään. Bluetooth-laitteen profiili tai profiilit kertovat, mitä käyttötarkoitusta varten laite on. Esimerkiksi autossa käytettävä hands free -kuulokesetti käyttää headset-profiilia (HSP) ja Nintendo Wii -ohjain käyttää vuorostaan human interface device -profiilia (HID). Jotta Bluetooth-laitteet voivat toimia yhdessä, tulee niiden molempien tukea samaa profiilia. Bluetooth-profiileja on kymmenittäin, mutta taulukossa 1 on esitelty niistä joitakin käytetyimpiä. (Jimblom n.d.)

TAULUKKO 1. Joitakin yleisimpiä Bluetooth-profiileja (Sony Electronic Inc. 2022, muokattu).

Profiili	Ominaisuudet	Käyttökohteet
a/v remote control profile (AVRCP)	a/v -laitteet etäohjaus	a/v-laitteet, kuulokkeet, puhelimet, tietokoneet ja ajoneuvoissa käytettävät laitteet
advanced audio distribution profile (A2DP)	musiikin striimaus	a/v-laitteet, kuulokkeet, puhelimet, tietokoneet ja ajoneuvoissa käytettävät laitteet
hands-free profile (HFP)	hands-free-kommunikointi	puhelimet, hands-free-kuulokesetit ja ajoneuvoissa käytettävät laitteet
headset profile (HSP)	hands-free-kuulokesetin ja kuulokkeiden kommunikointi	puhelimet, hands-free-kuulokesetit, ajoneuvoissa käytettävät laitteet ja tietokoneet
human interface device profile (HID)	langaton syötelaitteen yhteys	näppäimistöt, hiiret, puhelimet ja tietokoneet
serial port profile (SPP)	Bluetooth-laitteen käyttö virtuaalisena sarjaporttina	puhelimet, tietokoneet

2.2.5 Tietoturva

Tietoturvaan liittyvää asiaa käsiteltiin jonkin verran jo luvussa 2.2.2 Protokollat, jossa tarkasteltiin aihetta BLE-laitteiden parituksen osalta. Tässä luvussa perehdytään Bluetoothin tietoturvaan yleisemmällä tasolla.

Bluetoothin mahdolliset tietoturvahaavoittuvuudet voidaan jakaa kolmeen kategoriaan:

1. passiivinen salakuuntelu (passive eavesdropping)
2. man-in-the-middle-hyökkäykset (MITM)
3. identiteetin jäljitys (identity tracking). (Yang ym. 2019, 4)

Passiivinen salakuuntelu tarkoittaa sitä, että pahantahtoinen laite salakuuntelee Bluetooth-laitteiden välistä liikennettä ja pystyy lukemaan niiden välillä liikkuvaa dataa yleensä, koska se on saanut haltuunsa datan salauksessa käytetyn avaimen (Afaneh 2018, 74). BLE:n osalta tätä haavoittuvuutta vastaan on varauduttu Link Layerin tasolla tapahtuvalla AES-CCM-salauksella (Yang ym. 2019, 4).

MITM-hyökkäyksiä varten BLE suojautuu paritusvaiheessa. Tätä käsiteltiin luvussa 2.2.2 Protokollat, BLE:ä käsittelevässä osassa.

Identiteetin jäljitys tarkoittaa sitä, kun pahantahtoinen taho pystyy yhdistämään BLE-laitteen osoitteen tiettyyn käyttäjään ja voi näin ollen fyysisesti seurata tätä käyttäjää BLE-laitteen osoitteen perusteella. BLE vastaa tähän haasteeseen muuttamalla laitteen osoitetta määräajoin. (Yang ym. 2019, 5.)

Bluetoothiin liitetään myös seuraavat uhat:

- Bluejacking
- Bluesnarfing
- Bluebugging. (Marks 2022.)

Bluejacking tarkoittaa sitä, että pahantahtoinen taho pystyy yhdellä Bluetooth-laitteella kaappaamaan toisen Bluetooth-laitteen ja lähettämään viestejä kaapattuun laitteeseen. Tämä on enimmäkseen vain ärsyttävä ongelma, mutta

toisinaan Bluejacking-hyökkäys pitää sisällään phising-tietojenkalasteluviestejä, jotka voivat johtaa vakavampiin ongelmiin. (Marks 2022.)

Bluesnarfing on samantapainen hyökkäys kuin Bluejacking, mutta se ei tyydy ainoastaan lähettämään viestejä kaapattuun laitteeseen vaan se myös imee tietoja kaapatusta laitteesta. Nämä tiedot saattavat sisältää tekstiviestejä, kuvia, sähköpostiviestejä ja jopa identifioivia tietoja. (Marks 2022.)

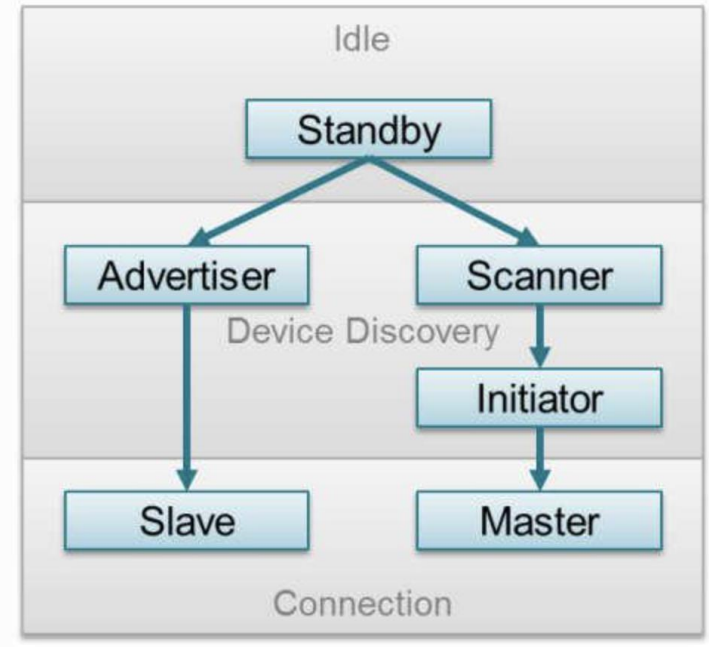
Bluebugging tarkoittaa sitä, että hyökkääjä saa avattua salaa Bluetooth-yhteyden käyttäjän kännykkään tai tietokoneeseen ja käyttää tätä yhteyttä avatakseen takaoven hyökkäyksen kohteena olevaan laitteeseen. Laitteeseen päästyään hyökkääjä voi tehdä siellä pahimmillaan mitä huvittaa. (Marks 2022.)

3 BLUETOOTH LOW ENERGYN YLEMMÄN TASON PROTOKOLLAT

BLE-protokollapinon Host-osan ylimmät kerrokset, Generic Access Profile (GAP), Attribute Protocol (ATT) ja Generic Access Attribute (GATT), muodostavat protokollat, joiden mukaan BLE-laiteet keskustelevat keskenään. Tämä luku keskittyy käsittelemään näitä protokollia, sillä nämä protokollat vaikuttavat suoraan Android-laitteelle kirjoitettavaan koodiin.

3.1 Generic Access Profile (GAP)

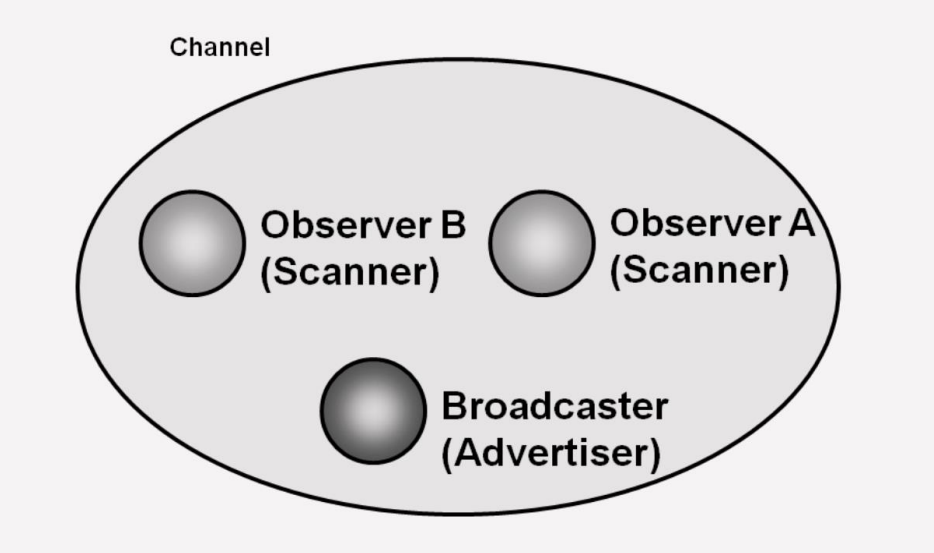
Generic Access Profile -taso vastaa yhteyden muodostamiseen liittyvistä toiminnoista. Taso hallinnoi saavutettavuuteen liittyviä proseduureja ja moodeja kuten laitteen havaitseminen (Device Discovery), linkin muodostaminen (Link Establishment), linkin katkaisu (Link Termination), turvallisuusominaisuuksien käynnistäminen (Initiation of Security Features) ja laitteen konfiguraatio (Device Configuration). Kuvio 12 osoittaa BLE-laitteen eri tilat riippuen laitteen roolista. (Texas Instrument 2016.)



KUVIO 12. BLE-laitteen Link Layer -tason ja GAP-tason tilojen diagrammi (Texas Instruments 2016).

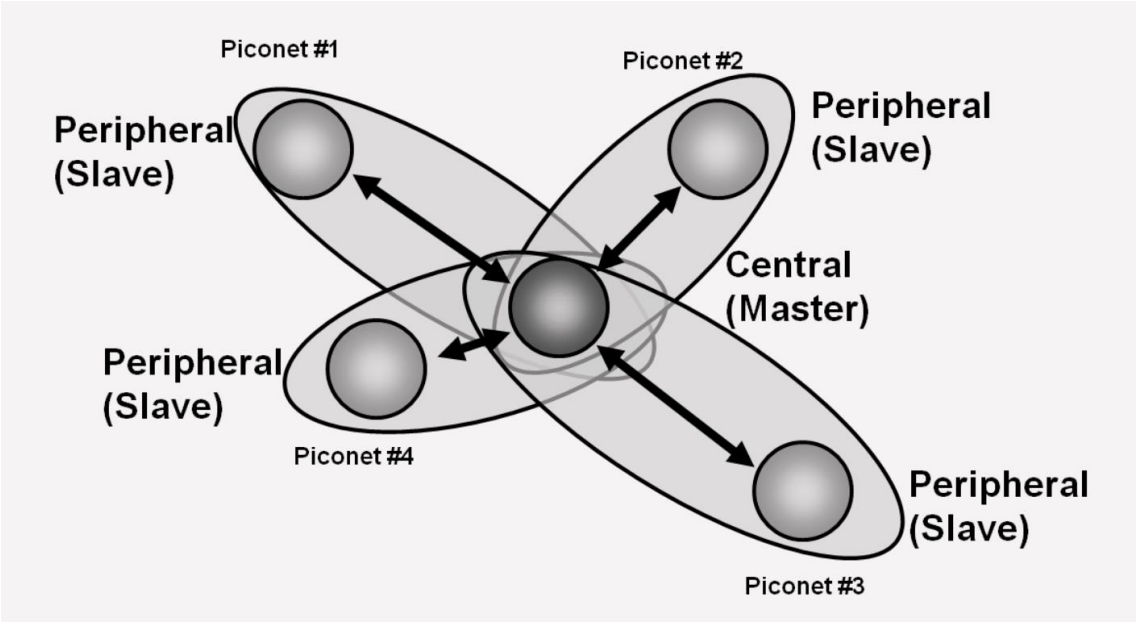
3.1.1 Laitteiden roolit

GAP määrittelee yhteyden muodostavien laitteiden roolit. Laitteiden roolit vaihtuvat sen mukaan missä vaiheessa yhteyden muodostusta prosessi on. Laitteen rooli riippuu yhteydenmuodostusvaiheen lisäksi myös siitä, mitä laite tekee Device Discovery -vaiheessa. Rooliin vaikuttaa siis se, onko laite Link Layer -tasolla Advertiser- vaiko Scanner-laite (kuvio 13), eli lähettääkö se itsestään tunnistedataa muille laitteille vai skannaako se kanavia löytääkseen etsimäänsä tunnistedataa. Itseään mainostava laite on GAP-tasolla, Device Discovery -vaiheessa, rooliltaan Broadcaster ja kanavia skannaava laite on vuorostaan Observer (kuvio 13). (Microchip, 2021.)



KUVIO 13. Device Discovery -vaiheen laiteroolit (Microchip 2021, muokattu).

Kun laitteet yltävät Connection-vaiheeseen (kuvio 14), tulee Broadcaster-laitteesta Link Layer -tasolla Slave-laite ja GAP-tasolla Peripheral-laite. Observer-laite ottaa vuorostaan Connection-vaiheessa rooleikseen Master (Link Layer -taso) ja Central (GAP-taso). (Microchip 2021.)

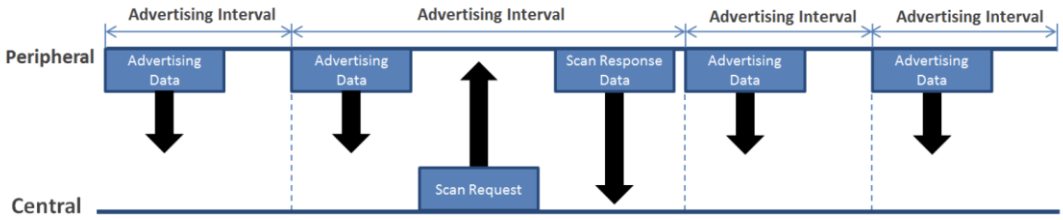


KUVIO 14. Connection-vaiheen laitteet (Microchip 2021, muokattu).

Townsendin mukaan (2022) GAP-tason roolien kannalta keskeisimmät konseptit ovat Peripheral ja Central. Hän kuvaa Peripheral-laitetta pieneksi, virrankulutukseltaan matalaksi ja resursseiltaan rajalliseksi laitteeksi, joka voi yhdistyä suuremman prosessointitehon Central-laitteeseen. Peripheral-laitteiksi hän nimeää esimerkiksi sykemittarit ja Central-laitteiksi älypuhelimet.

3.1.2 Advertising ja Scan Response Data eli BLE-laitteen lähettämä tunnistedata

Peripheral-laite voi lähettää Advertising-dataa kahdella eri tavalla. Nämä ovat Advertising Data Payload (mainoshyötykuorma) ja Scan Response Payload (skannausvastauksen hyötykuorma). Nämä hyötykuormat ovat keskenään muodoltaan ja mitaltaan identtisiä ja kumpikin voi siten sisältää 31 tavua dataa. Niiden ero on siinä, että ainoastaan Advertising Data on Peripheral-laitteelle pakollinen, sillä vain sitä lähetetään laitteelta johdonmukaisesti Central-laitteiden havaittavaksi. Scan Response lähetetään Peripheral-laitteelta vain ja vasta, jos Central-laite pyytää sitä (kuvio 15) ja Peripheral-laitteelle on sellainen määritelty. Tämän vapaaehtoisen hyötykuorman tarkoitus on toimittaa hieman lisää informaatiota, esimerkiksi merkkijono, joka kuvaa laitteen nimeä, Peripheral-laitteelta Central-laitteeelle. (Townsend 2022.)

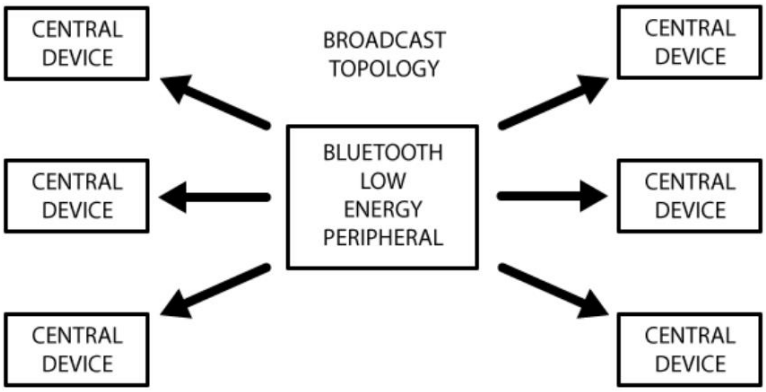


KUVIO 15. Tunnistedatan lähetysprosessi (Bui 2015).

Peripheral-laitteelle asetetaan siis tietty intervalli, jonka välein se lähettää Advertising-dataa. Pidempi intervalli säästää Peripheral-laitteen virtaa, mutta saa laitteen vaikuttamaan heikommin reagoivalta. Mikäli Central-laite haluaa lisätietoa Peripheral-laitteesta ennen yhteyden muodostusta, se voi lähettää Scan Requestin, eli kyselyn, Peripheral-laitteelle ja mikäli tälle on määritelty Scan Response -data, vastaa Peripheral-laite Central-laitteelle. Tämän jälkeen (ja muussa tapauksessa) Peripheral-laite jatkaa Advertising -datan lähettämistä asetetun intervallin mukaisesti, kuten kuviossa 15 on kuvattu. (Townsend 2022.)

3.1.3 Broadcast-verkon topologia

Bluetooth Low Energyn myötä esiteltiin Bluetooth-laitteille uusi verkkotopologia, Broadcast, joka toimii ainoastaan GAP-tason Device Discovery -vaiheessa, tarkalleen ottaen vain Link Layerin Advertising-vaiheessa. Broadcast-verkkoa käyttävät sellaiset BLE-laitteet, joiden tarkoitus on ainoastaan lähettää pieniä datamääriä lukuisille laitteille (kuvio 16). Tämä data pakataan joko Advertising Data -hyötykuormaan tai Scan Response -hyötykuormaan, minkä Central-laite lukee kanavia skannatessaan tai Scan Requestin lähetettyään. Tätä one-to-many-topologiaa ei voi yleensä hyödyntää enää siinä vaiheessa, kun laitteet ovat siirtyneet Connection-vaiheeseen, sillä yhteyden muodostamisen jälkeen Advertising-prosessi tyypillisesti katkeaa ja laitteet siirtyvät käyttämään GATT-protokollaa keskinäisessä kommunikaatiossaan. (Townsend 2022.)



KUVIO 16. Broadcast-topologia (Townsend 2022).

3.2 Attribute protocol (ATT)

Attribute Protocol (ATT) -tasolla laitteille jaetaan jälleen uudet roolit. Nämä eivät välttämättä vastaa GAP-tason Peripheral- ja Central-rooleja, sillä GAP- ja ATT-tason laiteroolit ovat toisistaan täysin riippumattomia. ATT-tason roolit ovat Server (palvelin) ja Client (asiakas) ja ne saattavat vaihtua laitteiden välillä koska tahansa, mutta laitteet edustavat aina keskenään eri roolia. Palvelinlaite on laitteista se, joka toimii hallinnoitavan datan tietokantana siis laitteena, josta luetaan ja jonne kirjoitetaan dataa. Asiakaslaite on vuorostaan se laite, joka suorittaa pyyntöjä ja / tai komentoja palvelinlaitteelle. (Hlapisi 2022.)

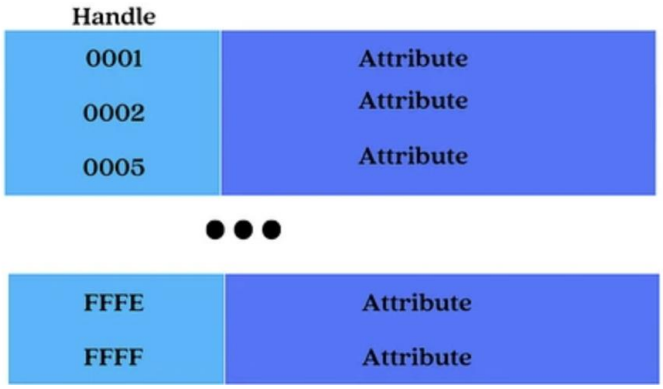
Android Developers (2021) antaa seuraavan esimerkin laitteiden vaihtuvista rooleista: älypuhelin ja aktiivisuusranneke muodostavat keskenään BLE-yhteyden. Kun aktiivisuusranneke lähettää puhelimeen keräämäänsä dataa, se toimii todennäköisesti palvelinlaitteena. Tilanteessa, jossa aktiivisuusranneke haluaa päivityksiä puhelimelta, toimii puhelin vuorostaan palvelinlaitteena. (Android Developers 2021.)

ATT-tason protokollat määrittelevät, kuinka palvelinlaite paljastaa dataa asiakaslaiteelle ja missä muodossa se sitä paljastaa. Se on BLE-laite, joka hyväksyy vertaiseltaan laitteelta sisään tulevat komennot, lähettää vastauksia, ilmoituksia (Notifications) ja viittauksia (Indications). Asiakaslaite toimii vuorostaan palvelinlaitteen käyttäjärajapintana. Asiakaslaitteella annetaan

komentoja ja pyyntöjä (Requests) palvelinlaitteelle, luetaan palvelinlaitteen lähettämään dataa ja hyväksytään palvelinlaitteelta tulevat ilmoitukset ja viittaukset. ATT-tason protokollat määrittelevät siis mitä dataa palvelinlaite luovuttaa asiakaslaitteelle asiakaslaitteen esittämien pyyntöjen pohjalta. (Afaneh 2018, 46.)

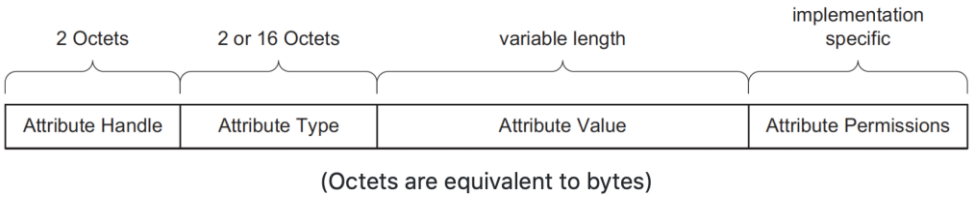
ATT-tasolla palvelinlaitteen säilömä tai keräämä, ja välittämä data on jäsennelty attribuuttitietokantaan (kuva 17). Attribuutti on geneerinen termi, jolla kuvataan minkä tahansa tyyppistä palvelinlaitteen asiakaslaitteelle välittämää dataa. Attribuutit rakentuvat aina seuraavasti:

- attribuuttityyppi (Universally Unique Identifier or UUID)
    - 16-bittinen (SIGiltä adoptoitu attribuutti) tai 128-bittinen (valmistajakohtainen attribuutti) numero
- attribuuttikahva (Handle)
    - 16-bittinen arvo, jonka palvelin asettaa jokaiselle attribuutille
    - ”osoite”, jonka perusteella asiakas voi kohdentaa toimensa nimetylle attribuutille
    - arvo on muuttumaton koko laitteiden välisen yhteyden ajan
- attribuutin luvat (Permissions)
    - määrittelevät saako attribuutille kirjoittaa ja saako sen sisältämän arvon lukea sekä voiko sille tehdä ilmoituksia tai viittauksia
    - määrittelevät turvallisuustasot kullekin yllä mainitulle toiminnalle
- attribuutin arvo (Value)
    - attribuutin sisältämä muuttuja-arvo. (Afaneh 2018, 47.)



KUVA 17. Kuvaus palvelinlaitteen attribuuttitietokannasta (Hlapisi 2022).

Kuvassa 18 on esitetty, millaisista osista attribuutti muodostuu. Kuvassa mainittu määre ’octet’ vastaa yhtä tavua eli kahdeksaa bittiä. Kuten edellä esitettiin, Attribute Typen pituus riippuu siitä käyttääkö attribuutti SIGin standardoimaa vai valmistajan itsensä määrittelemää UUID:ta.

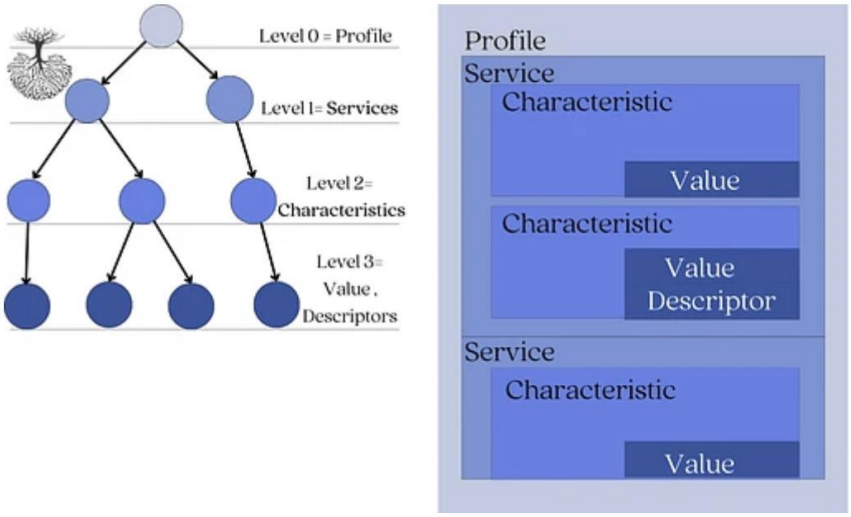


KUVA 18. Attribuutin looginen esitystapa (Bluetooth SIG Proprietary 2016, 2225).

3.3 Generic Attribute profile (GATT)

BLE-protokollapinon GATT-taso on palveluviitekehys (Service Framework), joka määrittelee aliproseduurit, joilla ATT-tason toimintoja säädellään. BLE-laitteet siirtyvät GATT-tasolle, kun ne ovat muodostaneet yhteyden keskenään. Tason protokollat vastaavat laitteiden välisestä kommunikaatiosta ja datansiirrosta. (Texas Instrument 2016.)

GATT-tasoa voi kuvata neljätasoisella puunmallisella viitekehyksellä (kuva 19). Puun juurisolmu (taso 0) on profiili (Profile). Seuraavalla tasolla (taso 1), eli profiilin lapsina, ovat palvelut (Services). Palveluiden lapsina (taso 2) ovat ominaisuudet (Characteristics). Ominaisuudet voidaan määritellä yksittäisellä arvolla (Value) tai arvolla ja siihen liittyvillä kuvaajilla (Descriptors), nämä muodostavat viitekehyksen alimman tason (taso 3). (Hlapisi 2022.)



KUVA 19. GATT-tason datan hierarkkinen rakenne (Hlapisi 2022).

Characteristics, ominaisuudet

Characteristic on perus tallennusyksikkö, johon sovelluksen dataa kirjoitetaan, tai josta sitä luetaan. Se vastaa tiedostoa tietokoneella. Jokaisen Characteristicin määrittelee aina Value (arvo) ja mahdollisesti Descriptor (kuvaus). Value sisältää sovellukseen tallennetun datayksikön. Vapaavalintaisesti käytettävä Descriptor pitää sisällään lisäinformaatiota siihen liittyen. Tällainen lisäinformaatio voi olla esimerkiksi ihmisluettava kuvaus arvosta, arvon hyväksytyt raja-arvot tai arvoon liittyvä määre. (Hlapisi 2022.)

Kukin Characteristic saa tunnisteekseen UUID:n. Tämä UUID on joko 16-bittinen virallisesti adoptoitu, SIGin määrittelemä, BLE-Chracteristic tai 128-bittinen BLE-laitteen valmistajan määrittelemä Characteristic-UUID. BLE-laitteen valmistaja voi hyödyntää siis SIGin standardoimia Chracteristicseja ja taata yhteensopivuuden saman profiilin laitteen ja sovelluksen välillä, tai päätyä käyttämään omaa kustomoitua Characteristiciaan ja näin varmistaa, että hänen valmistamaansa BLE-laitteen kanssa pystyy toimimaan vain hänen valmistamansa (tai valmistuttamansa) sovellus. (Townsend 2022.)

Services, palvelut

Service on pikemminkin datan organisointiin liittyvä yksikkö kuin tallennusyksikkö. Jos Characteristic on verrannollinen tiedostoon tietokoneella, vertautuu Service kansioon tai hakemistoon. Aivan kuten tietokoneella olla useita kansioita, voi BLE-laite myöskin sisältää enemmän kuin yhden Servicen. Samaan

analogiaan nojaten, kansio sisältää siihen liittyviä tiedostoja ja Service vastaavasti siihen liittyviä Chracteristicseja. (Hlapisi 2022.)

Jokainen Service saa tunnistusta varten oman UUID:n. Tämä voi olla joko 16-bittinen virallisesti adoptoitu SIG BLE Service -UUID tai 128-bittinen BLE-laitteen valmistajan oma kustomoitu Service-UUID. (Townsend 2022.)

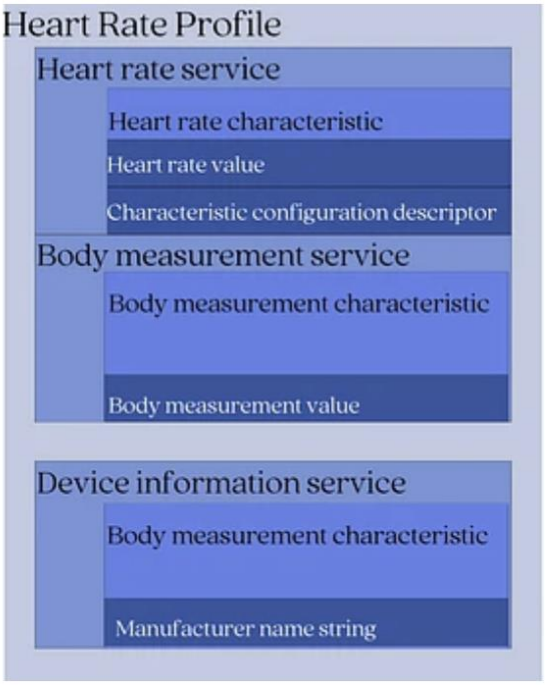
Profile, profiili

Profiili viittaa luvussa 2.2.4 Profiilit käsiteltyihin profiileihin. Profiili ei ole BLE-laitteella itsestään olemassa, vaan se valitaan ja asetetaan joko näistä SIGin ennalta määrittelemistä profiileista, tai BLE-laitteen valmistaja asettaa laitteelle luomansa oman profiilin (Townsend 2022).

Kun palvelinlaitteelle on määritelty jokin profiili, tulee sen toteuttaa tämän profiilin määrittelemät Servicet ja Characteristicsit. Jotkin profiilissa määritellyistä Serviceistä ja Chracteristicseista voivat olla vapaavalintaisia, mutta toiset niistä on pakko toteuttaa, mikäli palvelinlaite ilmoittaa olevansa yhdenmukainen tietyn profiilin kanssa. Tällä tavoin varmistetaan se, että saman GATT-profiilin omaavat BLE-laitteet pystyvät toimimaan keskenään. Esimerkiksi sykemittari, joka ilmoittaa olevansa yhdenmukainen GATT-profiilin Heart Rate kanssa tulee sisältää:

- 100 % varmuudella Heart rate -Servicen
- 100 % varmuudella Heart rate measurement -Characteristisin. (Hlapisi 2022.)

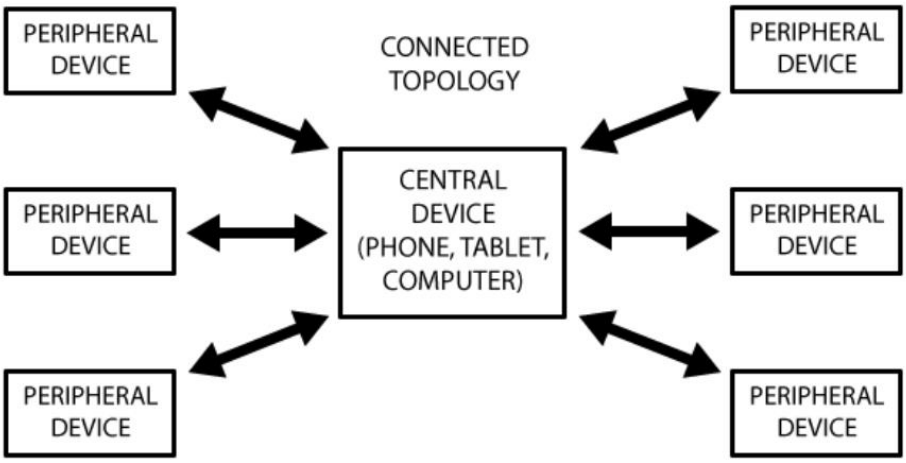
Yhtenäinen datan organisointi ja jaottelu tukee datan helppoa löytämistä, sen helppoa jakamista ja varmistaa laitteiden välisen keskinäisen toimintakyvyn (Hlapisi 2022). Kuvio 20 esittää kuvitteellisen sykemittarin rakenteen. Sykemittari ilmoittaa olevansa Heart Rate -profiilin kanssa yhdenmukainen, joten se sisältää Heart rate -Servicen (pakollinen), Body measurement -Servicen (vapaavalintainen) ja Device information -Servicen (pakollinen). (Hlapisi 2022.)



KUVIO 20. Kuvitteellisen sykemittarin profiilin hierarkkinen kuvaus (Hlapisi 2022).

3.3.1 Connected-verkon topologia

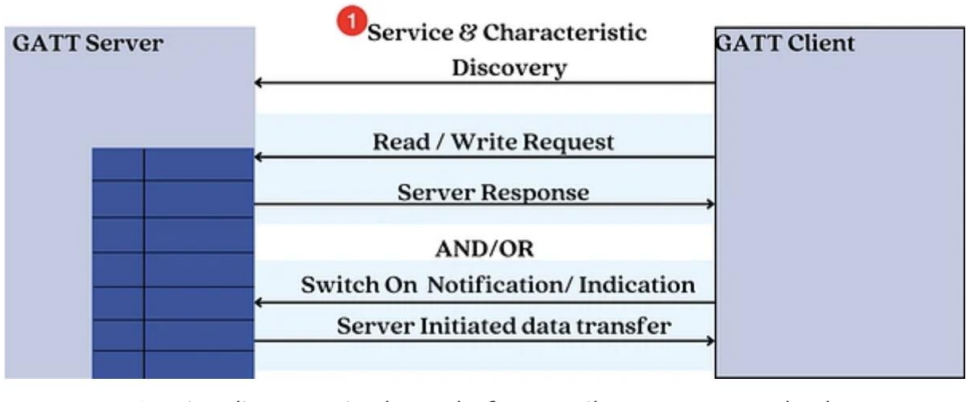
Peripheral-laite voi olla yhdistettynä vain yhteen Central-laitteeseen kerrallaan, mutta Central-laite voi olla yhteydessä usean Peripheral-laitteen kanssa saman aikaisesti. Jos kahden Peripheral-laitteen pitää välittää toisilleen dataa, on Central-laitteelle luotava kustomoitu viestilaatikko tätä toimintaa varten. Kummankin Peripheral-laitteen lähettämät viestit menevät aina tähän viestilaatikkoon, ja sieltä kumpikin noutaa itselleen osoitetut viestit. Central-laite pystyy viestimään kaikille siihen yhteydessä oleville Peripheral-laitteille kahdensuuntaisesti, eli se siis pystyy sekä lähettämään että vastaanottamaan dataa jokaiselta verkkoonsa kuuluvalta Peripheral-laiteelta (kuvio 21). (Townsend 2022.)



KUVIO 21. Connected-verkon topologia (Townsend 2022).

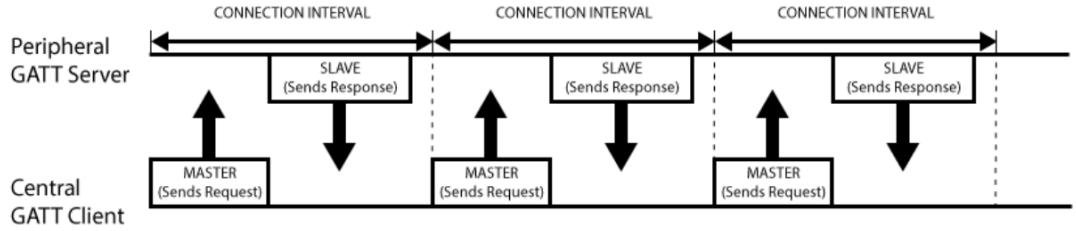
3.3.2 GATT-transaktiot

GATT-tason asiakaslaite on aina GATT-transaktioiden aloitteentekijä ja täten ensisijainen laite. Asiakaslaite aloittaa transaktion lähettämällä palvelinlaitteelle pyynnön (Request). Palvelinlaite vuorostaan vastaa pyyntöön (Response), ja on täten toissijainen laite. (Townsend 2022.) Yhteyden muodostamisen myötä asiakaslaite yleensä kartoittaa palvelinlaitteen sisältämät Servicet ja Chracteristicsit. Saatuaan tämän selvitettyä, asiakaslaite pystyy lukemaan ja kirjoittamaan tiettyjen Chracteristicsien arvoja, tai se pystyy pyytämään palvelinlaitetta ilmoittamaan (Notification), mikäli jokin sen Characteristics Value muuttuu (kuvio 22). (Hlapisi 2022.) Palvelinlaiteelle on määritelty eri Servicejen ja Characteristicsien luku- ja kirjoitusoikeudet.



KUVIO 22. GATT-transaktion etenemien, Service & Characteristic Discovery on aina aloittava vaihe (Hlapisi 2022).

Townsendin (2022) mukaan GATT-transaktioiden aivan alussa, kun laiteiden välinen yhteys on muodostunut, Peripheral-laite ehdottaa Central-laiteelle yhteysintervallia, jonka myötä Central-laite yrittää uudelleen yhdistyä Peripheral-laitteeseen ehdotetun intervallin mukaisesti nähdäkseen onko Peripheral-laitteen sisältämä data päivittynyt (kuvio 23). (Townsend 2022.)



KUVIO 23. GATT-transaktio yhteysintervallin näkökulmasta (Townsend 2022).

4 ANDROID-MOBIILILAITE JA BLE-YHTEYS

4.1 Android

Android on mobiililaitteiden käyttöjärjestelmä, jonka kehitystyö alkoi lokakuussa 2003 Android Inc:in toimesta. Vuonna 2005 Google osti tämän Androidia kehittäneen yrityksen, ja samoihin aikoihin Androidin perustaksi päätettiin ottaa Linux. Tämän päätöksen takana oli ajatus avoimen lähdekoodin (open source) käyttöjärjestelmästä. Tämän oli määrä helpottaa käyttöjärjestelmän päätymistä kolmannen osapuolen mobiililaitteisiin. Google ja sen Android-tiimi kokivat, että yritys hyötyisi taloudellisesti, jos se pystyisi tarjoamaan laajasti palveluita ja sovelluksia. Androidin logon suunnitteli Irina Blok, ja yhdessä Googlen kanssa hän päätti julkaista logon Creative Commons 3.0 -lisenssillä. Tämän ansiosta logostakin tuli tietyin ehdoin vapaasti muokattava open source -projekti. (Callaham 2022.)

Ensimmäinen Android-käyttöjärjestelmällä varustettu puhelin, T-Mobile G1, tuli markkinoille lokakuussa 2008. Vaikka puhelin itsesään sai heikot arviot, sisälsi se useita Googlen eri palveluita, kuten Google Mapsin, YouTuben, Googlen silloisen web-selaimen ja Googlen sovelluskaupan ensimmäisen version. Googlen suunnitelma mobiililaitteiden varalle alkoi piirtyä esiin myös kuluttajille. (Callaham 2022.)

Android OS on vuosien ja versiopäivitysten myötä muuttunut alkuajoistaan merkittävästi niin UI:n (käyttäjärajapinta), UX:n (käytettävyys) kuin ominaisuuksiensakin osalta. Liitteessä 2 on esitelty suurimmat versiopäivitykset ja joitakin niiden mukanaan tuomista ominaisuuksista ja muista muutoksista. Tällä hetkellä tuorein Android OS -versio on Android 13, koodinimeltään Tiramisu. Jälkiruokiin viitanneet koodinimet otettiin Androidilla käyttöön OS-versiosta 1.5 (Cupcake) alkaen. (Callaham 2022.)

4.2 Android BLE API

Android BLE API tarkoittaa Androidin Bluetooth Low Energy -ohjelmointirajapintaa. Android tarjoaa sisäänrakennetun tuen BLE Central-laitteen roolille ja useampia rajapintoja, joiden avulla sovellukset voivat löytää BLE-laitteita, tehdä Service-kyselyjä ja siirtää dataa (Android Developers 2021).

Welie (2019) toteaa, että Androidille suunnattujen BLE-sovellusten kirjoittaminen on merkittävän haastavaa ja monimutkaista, sillä Googlen tarjoama dokumentaatio ei ole tasoltaan kummoista. Se sisältää paikoin vajavaista tai vanhentunutta tietoa eivätkä sen tarjoamat sovellusesimerkit myöskään kerro riittävällä tarkkuudella, miten BLE:n saa toimimaan oikeaoppisesti. (Welie 2019.) Ong (2020) kirjoittaa niin ikään, että Android BLE APIt ovat täynnä dokumentoimattomia sudenkuoppia.

Welie (2019) listaa muiksi ongelmakohdiksi seuraavat asiat:

- Android BLE APIt toimivat Androidin protokollapinon alemmilla kerroksilla, ja useimmat sovellukset tarvitsevat joitakin kerroksia niiden päälle helppokäyttöisyyden nimissä. Hän kaipaa Androidille samantyyppistä ratkaisua kuin Applen iOS-käyttöjärjestelmälle laadittu CoreBluetooth, joka huolehtii alemman tason tehtävistä, kuten esimerkiksi komentojen jonotuksesta, pysyvän yhteyden muodostuksesta ja yhteyden hallinnasta.
- Puhelinvalmistajat tekevät muutoksia oletusarvoiseen BLE-pinoon, tai korvaavat osia siitä omilla implementaatioillaan. Näin ollen yhdellä puhelimella toimiva BLE-sovellus ei välttämättä toimikaan toisen valmistajan puhelimella.
- Etenkin Androidin vanhemmilla versioilla, versiot 4, 5 ja 6, löytyy paljon tunnettuja ja tuntemattomia virheitä Androidin omassa koodissa, jotka täytyy jotenkin käsitellä, jotta sovelluksen saa toimimaan vakaasti. (Welie 2019.)

Nämä Android BLE APIen haasteet kävivät selväksi opinnäytetyön taustalla olevan projektin yhteydessä. Projektin tarkoituksena oli virheiden korjaus ja uusien toimintojen kehitys BLE Peripheral-laitteita hallinoivan sovelluksen käyttämässä kirjastossa. Toimeksiantajayritys oli aiemmin uudelleen kirjoittanut

tämän kirjaston natiiveiksi, eli alustakohtaisiksi, kirjastoiksi Android- ja iOS-laitteille alustariippumattoman alkuperäiskirjaston lähdekoodin pohjalta.

4.2.1 Projektin perustamiseen liittyvät huomiot

Ong (2019) painottaa, että Android-BLE-projektia perustettaessa on syytä asettaa alimmaksi APIn kohdetasoksi 21 (minimum target API), jonka vastine Android versioissa on 5.0. Syy tähän on se, että Android 5.0 myötä Google paransi Androidin BLE-toiminnallisuutta merkittävästi kehittäjäystävällisempään suuntaan. Androidille esiteltiin ohjelmointirajapinnat BluetoothLeScanner ja ScanFilter ja APIen toiminnasta tuli muutenkin monin tavoin vakaampaa. Suositeltu ohjelmointikieli on Kotlin. (Ong 2019.)

Opinnäytetyön taustalla olevassa projektissa alin APIn kohdetaso oli 21 ja projektin kielenä käytettiin Kotlinia. Projektia työstettiin Android Studiolla ja projektissa keskityttiin nimenomaan Android- ja BLE-laitteen välistä kommunikaatiota hallinnoivan kirjaston ohjelmointiin. Tässä projektissa ja kirjastossa voitiin katsoa, että Android-laite edustaa Central- / Client-laitetta ja projektin asiakkaan toimittamat laitteet olivat BLE-termein Peripheral- / Server-laitteita.

4.2.2 Lupien käsittely

Projektin käynnistysvaiheessa on syytä kiinnittää huomiota tarvittavien lupien käsittelyyn (Permission Handling). Android-projekteissa luvilla tarkoitetaan niitä käyttölupia, joita Android-laite pyytää käyttäjältä tämän avatessa sovelluksen. Luvat määritellään projektin AndroidManifest.xml-tiedostossa (kuva 24).

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

KUVA 24. Lupien määritys AndroidManifest.xml-tiedostossa.

Ong (2020) kuvaa tarvittavat luvat tapauksessa, jossa alin API kohdetaso on 21, seuraavasti:

- BLUETOOTH antaa sovellukselle luvan muodostaa yhteyden Bluetooth-laitteen kanssa.
- BLUETOOTH_ADMIN antaa sovellukselle luvan skannata Bluetooth-laitteita pysyvän yhteyden muodostusta varten.
- ACCESS_FINE_LOCATION vaaditaan Android 6.0 -versiosta alkaen, jotta skannaustulokset tulevat näkyviin. BLE-skannaus voi tahattomasti paljastaa käyttäjän sijainnin kehittäjille, jotka skannaavat tiettyjä BLE-majakoita, joten tämä tarkan sijainnin lupa on käyttäjänsuojan nimissä vaadittu. Vaikka ACCESS_FINE_LOCATION-lupa ei olekaan vaadittu kaikilla Android-versioilla, on suositeltavaa käyttää sitä, sillä suuri osa Android-laitteista sen vaatii.

Edellä mainituista luvista ainoastaan sijaintiin liittyvä lupa lasketaan Android-termein vaaralliseksi luvaksi. Tämä tarkoittaa sitä, lupa on yksiselitteisesti ja selkeästi pyydettävä käyttäjältä. Muut tässä yhteydessä mainitut luvat ovat sellaisia, ettei niihin tarvita käyttäjältä erillistä suostumusta, vaan ne hyväksytään sovelluksen asennuksen yhteydessä. (Ong 2020.)

Käyttäjärajapinnassa tulee tehdä Android-laitteen Bluetoothin päällä olo -tarkistus sekä käsitellä ACCESS_FINE_LOCATION-lupaan liittyvä käyttäjän antama vastaus. Android BLE -sovellus ei voi toimia ilman päällä olevaa Bluetoothia eikä skannaus tuota vastauksia ilman lupaa tarkkaan sijaintiin. (Ong 2020.)

Android Developers (2022) kertoo, että mikäli alin APIn kohdetaso on 31 tai enemmän, tulee AndroidManifest.xml-tiedostossa määritellä seuraavat luvat:

- BLUETOOTH_SCAN
- BLUETOOTH_ADVERTISE
- BLUETOOTH_CONNECT
- ACCESS_FINE_LOCATION

Yllä esitetyt luvat parantavat käyttäjän turvallisuutta. Niillä määritellään Android-laitteen näkyvyys, laitteen tarkan sijainnin näkyminen sekä yhteyden

muodostaminen valmiiksi paritettujen laitteiden kanssa. (Android Developers 2022.)

4.2.3 Peripheral-laitteiden skannaus

Ennen kuin BLE-laitteeseen voi muodostaa yhteyden, tulee kyseinen laite löytää BLE-skannauksen avulla. Skannaus suoritetaan BluetoothLeScannerilla, joka on BluetoothAdaterin objekti. (Welie 2019.) BluetoothAdapter on vuorostaan Android-laitteen Bluetooth-raudan ilmentymä. Se tarjoaa informaatiota tämän Bluetooth-raudan on/off-tilasta, mahdollistaa pysyvän yhteyden BLE-laiteyhteyksille tehdyt kyselyt ja BLE-skannauksen aloituksen. (Ong 2020.)

Skannaus aloitetaan kutsumalla BluetoothLeScannerin startScan-metodia. Skannaukselle voidaan määritellä filttereitä ScanFilterin avulla, mutta tämä ei ole välttämätöntä. Filtteröinnillä voidaan rajata skannauksessa löytyviä laitetyyppejä. (Ong 2020.) Skannauksen filtteröinti voidaan suorittaa joko BLE Services -UUID:lla, laitteen nimellä (kuva 25) tai hyväksyttävän laitteen MAC-osoitteella. Filtterit tulee määritellä ennen skannauksen aloittamista. (Welie 2019.)

```
private var deviceBrands = listOf("Jabra Evolve2 65", "JBL Charge 3")
private var filters: List<ScanFilter> = deviceBrands.map { ScanFilter.Builder().setDeviceName(it).build() }
```

KUVA 25. Skannauksen filttereiden luonti.

Tulosten filtteröinnin lisäksi skannaukselle voidaan määritellä myös asetuksia, joiden mukaan skannaus suoritetaan. Asetukset määritellään ScanSetting.Builder-luokkaa kutsumalla (kuva 26). Ne on määriteltävä ennen kuin skannaus käynnistetään. Asetuksia, joita Android sallii säädettävän ovat:

- BLE Scan mode
    - SCAN_MODE_BALANCED
    - SCAN_MODE_LOW_LATENCY
    - SCAN_MODE_LOW_POWER
- Callback-funktion tyyppi Advertisement-datan perusteella
    - CALLBACK_TYPE_ALL_MATCHES
    - CALLBACK_TYPE_FIRST_MATCH
    - CALLBACK_TYPE_MATCH_LOST
- Kynnysarvo Advertisement-datan ilmestymisajalle
    - MATCH_MODE_STICKY
    - MATCH_MODE_AGGRESSIVE (Ong 2020.)

```
private var settings: ScanSettings = ScanSettings.Builder()
    .setScanMode(ScanSettings.SCAN_MODE_LOW_POWER)
    .build()
```

KUVA 26. Skannauksen asetusten määrittely.

Skannausta käynnistettäessä on hyvä tarkistaa, onko skannaus jo käynnissä ja pysäyttää se ja käynnistää uudelleen, mikäli tämä on tilanne. Kuvassa 27 on esitetty, miten skannauksen käynnistyksen voi esimerkiksi hoitaa.

Kuvasta 27 käy ilmi, että BluetoothLeScanner.startScan-funktio ottaa parametreikseen ”filters, settings, savedScanCallback”. Edellä on kuvattu funktion kaksi ensimmäistä parametria. Funktion viimeinen parametri, ”savedScanCallback”, on funktioparametri, eli muuttujan sijasta parametrina on funktio. Kun koodiblokkia tutkii syvemmin, huomaa, että ”savedScanCallback” saa ylempänä arvokseen ”ValveScanCallback”-luokan (kuva 28) instanssin, jolle on syötetty parametriksi startScan-apufunktion parametrinaan saama ”scanCallback”, joka vuorostaan edustaa niin ikään funktioparametria. Kun skannaus tuottaa annettujen filttereiden ja asetusten mukaisesti tuloksen, eli löytää annettuja arvoja vastaavan Peripheral-laitteen, se kutsuu ”savedScanCallback”-funktiota.

```
/**
 * Start scanning for compatible BLE devices. If called multiple times without a call to
 * [stopScan], stops the previous scan and starts a new one with the given callback.
 *
 * @param[scanCallback] called on each discovered compatible device
 */
fun startScan(scanCallback: ScanCallback) {
    if (savedScanCallback != null) {
        btScanner!!.stopScan(savedScanCallback)
    }
    savedScanCallback = ValveScanCallback(scanCallback)

    if (btAdapter == null) {
        btAdapter = BluetoothAdapter.getDefaultAdapter()
    }
    btScanner = btAdapter!!.bluetoothLeScanner

    btScanner!!.startScan(filters, settings, savedScanCallback)
}
```

KUVA 27. Apufunktio startScan, jolla varmistetaan, ettei skannauksia yritetä ajaa päällekkäin.

Kuvassa 28 ”ValveScanCallback”-luokassa kutsutaan abstraktia ”android.bluetooth.le.ScanCallback”-luokkaa, jolta löytyy skannaustulosten käsittelyyn metodeja, joita voi tarpeensa mukaan ylikirjoittaa. Kuvassa esitetyssä tapauksessa on ylikirjoitettu luokan metodi ”onScanResult”, joka viimeisellä rivillään kutsuu parametriksi saamansa ”savedScanCallback”-funktion sisäistä funktiota, jonka nimi on niin ikään ”onScanResult”. Tässä tapauksessa ”savedScanCallback”-funktio löytyy käyttäjärajapintaa varten kirjoitetusta ”MainActivity”-moduulista, joka hyödyntää niin ikään samaista abstraktia ”android.bluetooth.le.ScanCallback”-luokkaa ja sen metodia ”onScanResult”.

```
class ValveScanCallback constructor(val savedScanCallback: ScanCallback) :
    ScanCallback() {

    override fun onScanResult(callbackType: Int, result: ScanResult?) {
        val TAG = "ValveScanCallback"
        super.onScanResult(callbackType, result)
        Log.d(TAG, msg: "Found compatible device '${result?.device?.name}'")
        savedScanCallback.onScanResult(callbackType, result)
    }
}
```

KUVA 28. ValveScanCallback-luokka.

Tämä callbackien muodostama ketju kuulostaa monimutkaiselta ja sitä se onkin. Ong (2020) kuitenkin kirjoittaa, että BLE-kehittämisessä on tärkeää muistaa laittaa toiminnot jonoon ja odottaa aina edellisen callbackin toteutumista ennen kuin seuraavaan operaatioon ryhdytään. Hän painottaa, ettei tämä kosketa ainoastaan luku- ja kirjoitusoperaatioita, vaan ihan kaikkia BLE-operaatioita.

Skannaus pitää saada katkaistua koska tahansa myös suoralla komennolla. Käyttäjälle on syytä tarjota mahdollisuus katkaista aloitettu skannaus koska tahansa, joten tällaista komentoa varten kannattaa luoda funktio. Skannaus on kannattavaa lopettaa myös siinä tilanteessa, kun käyttäjä päättää muodostaa yhteyden Peripheral-laitteen kanssa (Ong 2020). Kuvassa 29 esitetty stopScan-funktio on apufunktio, joka nimensä mukaisesti kutsuu BluetoothLeScanner.stopScan-funktiota ja täten pysäyttää skannauksen.

```
/**
 * Stop scanning. If not already scanning, does nothing.
 */
fun stopScan() {
    if (savedScanCallback != null) {
        btScanner!!.stopScan(savedScanCallback)
        savedScanCallback = null
    }
}
```

KUVA 29. Apufunktio stopScan.

4.2.4 Yhteyden muodostaminen Peripheral-laitteen kanssa

Käytännössä Peripheral-laitteisiin on mahdollista muodostaa yhteys kahdella eri tavalla, joko yhteys muodostetaan automaattisesti tai se muodostetaan manuaalisesti. Manuaalinen yhteyden muodostus tarkoittaa sitä, että käyttäjä valitsee skannauksen tuloksista laitteen, johon haluaa muodostaa yhteyden Android-laitteellaan. Automaattinen yhteyden muodostus tapahtuu vuorostaan itsestään, ilman käyttäjän syötettä. Automaattista yhteyden muodostusta käytetään ainoastaan tapauksissa, jolloin tiedetään tarkasti Peripheral-laitteen lähettämän Advertisement-datan perusteella, että kyseessä on haluttu Peripheral-laite. (Ong 2020.)

BLE-laitteilla automaattinen yhteyden muodostus on riippuvainen seuraavista asioista:

- Löytyykö ennakkoon määritelty Peripheral-laite skannauksella tavoitetuista Advertisement-datapaketeista?
- Onko Android-laitteen ja Peripheral-laitteen välille muodostettu pysyvä yhteys (bonding)?
- Löytyykö Peripheral-laite Android-laitteen välimuistista (cache)? (Welie 2019)

Yhteyden muodostus tapahtuu Android Developers (2021) mukaan siten, että Android-laite muodostaa yhteyden GATT-serveriin, eli Peripheral-laitteeseen, connectGatt-metodin avulla (kuva 30). Metodi ottaa kolme parametria, jotka ovat Androidin Context-objekti, autoConnect-Boolean-arvon sekä referenssin abstraktiin BluetoothGattCallback-luokkaan. Metodi connectGatt palauttaa

BluetoothGatt-luokan instanssin Peripheral-laitteelta. Tämän instanssin avulla Android-laite voi suorittaa Client-laitteen toimintoja. (Android Developers 2021.)

Kuvassa 30 esitetty connectGatt-kutsu on yksi monista vaihtoehtoisista tavoista kutsua kyseistä metodia. Tämä on suositeltu vaihtoehto, kun projektin spesifikaatioissa on rajattu alimmaksi API kohdetasoksi 21. (Ong 2020).

KUVA 30. BLE-yhteyden muodostaminen (Android Developers 2021).

Metodin connectGatt viimeinen parametri, joka sisältää referenssin abstraktiin BluetoothGattCallback-luokkaan, on callback, jota käytetään jatkossa Peripheral-laitekohtaisten operaatioiden suorittamiseen. Näitä operaatioita ovat esimerkiksi Peripheral-laitteelle kirjoittaminen ja siltä lukeminen. (Welie 2019.) Tämän callbackin referenssi on Ongin (2020) mukaan nimenomaan se, millä Client-laitteen (Android) suorittamia operaatioita välitetään.

BluetoothGattCallback on abstrakti luokka, mikä tarkoittaa sitä, että se sisältää metodeja, joita ylikirjoittamalla Android-laite saa ilmoituksia (notifactions) BluetoothGatt-liitännäisistä callbackeista. Keskeisin näistä metodeista on yhteyden muodostuksen kannalta onConnectionStateChange-metodi, joka tuottaa tärkeää tietoa BLE-yhteyden tilasta. Onnistunut yhteyden muodostus asettaa mainitun metodin status-parametrin arvoksi GATT_SUCCESS ja newState-parametrin arvoksi BluetoothProfile.STATE_CONNECTED. (Ong 2020.)

Opinnäytetyön taustalla oleva projekti hyödynsi Polidean BLE-rajapintakirjastoa nimeltä RxAndroidBle2, mistä syystä näissä projektista noukituissa esimerkkikoodeissa näkyy tuon kirjaston tarjoamia metodeja ja muita toiminnallisuuksia. Projektissa käytetty BLE-rajapintakirjaston käytöllä on pyritty helpottamaan Android BLE APIen asettamia haasteita. Se ei ole ainoa BLE-

rajapintakirjasto, mutta Polidea (2016) kirjoittaa, että sen kirjasto tuo helpotuksen lukuisten ja taas lukuisten asynkronisten callbackien, numeraalisten statusarvojen ja säikeiden hallintaan.

Seuraavissa kuvissa on esitetty, miten projektissa muodostettiin yhteys (kuva 31) Peripheral-laitteeseen, miten yhteyden tilaa valvottiin (kuva 32) ja miten yhteys katkaistiin (kuva 33). Projektin asiakkaaseen tai tämän tuotteisiin osoittavat viittaukset on poistettu tai korvattu anonyymiksi jättävällä vaihtoehdolla tässä ja muissa projektin koodipohjasta poimituissa kuvissa.

```
/**
* Connects to a device.
*
* Also reads all parameters to enable user to get values of them.
* Broadcast receives ACTION_GATT_CONNECTED.
*/
fun connect(peripheral: examplePeripheral): Boolean {

    examplePeripheral = peripheral
    val macAddress = peripheral.btDevice.address

    if (device == null ||
        device!!.connectionState == RxBleConnectionState.DISCONNECTED
    ) {
        device = rxBleClient.getBleDevice(macAddress)
        connectionObservable = prepareConnectionObservable()

        connectionObservable
            .delay(delayAmount, TimeUnit.MILLISECONDS, Schedulers.computation())
            .flatMapSingle { it: RxBleConnection
                it.discoverServices() }
            .delay(delayAmount, TimeUnit.MILLISECONDS, Schedulers.computation())
            .observeOn(AndroidSchedulers.mainThread())
            .delay(delayAmount, TimeUnit.MILLISECONDS, Schedulers.computation())
            .subscribe({ it: RxBleDeviceServices!
                onServicesDiscovered(it) }, { logError(it) })
            .let { it: Disposable!
                disposable.add(it) }
        observeConnectionState()
    }
    return true
}
```

KUVA 31. Yhteyden muodostus ja connectionObservable, joka tarkkailee yhteyden ja löytyneiden Gatt-Servicejen tilaa.

ConnectionObservablen sisällä jokaisen vaiheen välissä oli viivettä (delay), sillä Peripheral-laitteet toimittanut asiakas ilmoitti, että he olivat omissa testeissään alkuperäisen alustariippumattoman sovelluksen yhteydessä havainneet, että Peripheral-laitteiden luku toimi vakaammin, kun vaiheesta toiseen siirtymistä

hidastettiin viiveen avulla. Welie (2019) huomauttaa aiheeseen liittyen, että discoverServices-funktiokutsu laukaisee joukon matalan protokollatason komentoja, jotka vievät aikaa tyypillisesti sekunnin tai enemmän, riippuen Peripheral-laitteella olevien Gatt-Servicejen, Gatt-Characteristicsien ja Descriptorien määrästä.

Kuvassa 32 on kuvattu sekä prepareConnectionObservable-funktio, joka suorittaa yhteyden muodostuksen ja valvoo sen tilaa, että observeConnectionState-funktio, joka tarkkailee yhteyden tilan lippuja. Mikäli tilalippu ilmaisee, että yhteys on muodostettu onnistuneesti, kutsuu se parametria getParametersFromCharacteristics, joka vuorostaan lukee Peripheral-laitteen Gatt-Characteristicsien sisältämän datan siltä osin kuin niihin kirjatut luvat sen antavat. Jos tilalippu ilmaisee, että yhteys on katkaistu onnistuneesti, kutsuu se funktiota disconnect (kuva 33), joka huolehtii, että yhteyden katkaisusta lähtee tieto myös käyttäjärajapintaan.

```
private fun prepareConnectionObservable(): Observable<RxBleConnection> =
    device!!.establishConnection( autoConnect: false)
        .takeUntil(disconnectTriggerSubject)
        .compose(ReplayingShare.instance())

private fun observeConnectionState() {

    device?.observeConnectionStateChanges()
        ?.delay(delayAmount, TimeUnit.MILLISECONDS, Schedulers.computation())
        ?.subscribe(
            { it: RxBleConnection.RxBleConnectionState!
                if (it == RxBleConnectionState.CONNECTED) {
                    getParametersFromCharacteristics()
                }
                if (it == RxBleConnectionState.DISCONNECTED) {
                    disconnect()
                }
            },
            { logError(it) })
        ?.let { disposable.add(it) }
}
```

KUVA 32. Yhteyden muodostus ja tilan valvonta.

```
/**
 * Disconnects device from the manager.
 * After disconnect is done successfully broadcast receives ACTION_GATT_DISCONNECTED.
 */
fun disconnect() {
    triggerDisconnect()
    broadcastUpdate(ACTION_GATT_DISCONNECTED)
}

private fun triggerDisconnect() = disconnectTriggerSubject.onNext(Unit)
```

KUVA 33. Yhteyden katkaisu.

4.2.5 Gatt-Servicejen kartoitus

Gatt-Servicejen kartoitus on käytännössä pakollinen seuraava askel sen jälkeen, kun BLE-laitteiden välinen yhteys on onnistuneesti muodostettu. Jotta Android-laitteella voidaan suorittaa operaatioita Peripheral-laitteelle, tulee Android-laitteen tuntea Peripheral-laitteen datarakenne. Ong (2020) kuvaa sen merkitystä analogialla kartan luvusta vieraassa paikassa. Kuten luvussa 3.3 Generic Attribute (GATT) todettiin, yksittäinen Gatt-Service on kuin kansio, joka sisältää tiedostoja. Nämä tiedostot ovat Gatt-Characteristicseja. Ne vuorostaan sisältävät attribuutteja, joita kuvaavat Valuet ja Descriptorit.

RxAndroidBle2-kirjasto hoitaa Gatt-Servicejen löytämisen funktiolla discoverServices (kuva 31) ja kun Gatt-Servicet on saatu selville, jatkuu connectionObservablen prosessi siten, että se kutsuu funktiota onServicesDiscovered, joka ottaa parametrikseen aiemmassa vaiheessa havaitut Gatt-Servicet (kuva 34). Funktio onServicesDiscovered asettaa löydetyt Gatt-Servicet talteen luokkamuuttujaan ja nostaa lipun ACTION_GATT_CONNECTED, joka ilmoittaa, että yhteys Gatt-Serveriin, eli Peripheral-laitteeseen, on muodostettu.

```
private fun onServicesDiscovered(rxBleDeviceServices: RxBleDeviceServices) {
    services = rxBleDeviceServices
    broadcastUpdate(ACTION_GATT_CONNECTED)
}
```

KUVA 34. Funktio onServicesDiscovered.

4.2.6 Luku- ja kirjoitusoperaatioiden suorittaminen

Android BLE APIen luku- ja kirjoitusoperaatiot ovat asynkronisia, mikä tarkoittaa sitä, että lähettyyn komentoon tulee vastaus välittömästi, mutta data, jota komennolla luetaan tai kirjoitetaan, liikkuu hitaammin callback-funktion välityksellä. Lisäksi Androidin BluetoothGatt-lähdekoodiin on tehty toiminnallisuus, joka estää ajamasta useampaa kuin yhtä asynkronista operaatiota kerrallaan. Koska asynkronisia operaatioita voi ajaa vain yhden kerrallaan, on ennen uuden operaation käynnistämistä odotettava aina, että aiemmin aloitetun operaation callback-funktio on ensin ajettu loppuun. (Welie 2019.)

Ratkaisuna yksi kerrallaan -sääntöön Welie (2019) kertoo olevan ainoastaan komentojen jonottamisen. Hän perustelee väitettään sillä, että riippumatta sovelluksesta, hänen tutkimansa Android BLE APIen hallintaa helpottamaan laaditut kirjastot käyttävät järjestelmällisesti komentojen jonotusta luku- ja kirjoitusoperaatioissaan. Ong (2020) mainitsi samasta asiasta kirjoittaessaan toimintojen jonottamisesta.

Asynkronisuuden lisäksi BluetoothGattCharacteristicsien luku- ja kirjoitusoperaatioissa tulee ottaa huomioon, että nämä operaatiot onnistuvat ainoastaan, jos ne on määritelty kirjoitettaviksi ja luettaviksi. Mikäli luku- tai kirjoitusoperaatio ei ole sallittu, se kaatuu virheeseen GATT_READ_NOT_PERMITED- tai GATT_WRITE_NOT_PERMITED-virheeseen. (Ong 2020.)

BluetoothGattCharacterisitic sisältää getProperties-metodin, jonka avulla voi selvittää salliiko kyseinen characteristic luku- tai kirjoitus- tai luku- ja kirjoitusoperaatioita (Ong 2020). Opinnäytetyön taustalla olevassa projektissa luku- ja kirjoitusoperaatiot hoidettiin hyödyntämällä Polidean RxAndroidBle2-kirjaston metodeja readCharacteristic (kuva 36) ja writeCharacteristic (kuva 37).

```
/**
 * Reads given GattCharacteristic by using RxBleConnection method readCharacteristic(UUID) and
 * broadcasts characteristic update after gaining data from the read operation.
 *
 * @param characteristic    GattCharacteristic to be read.
 */
fun readCharacteristic(characteristic: ExampleGattCharacteristic) {
    connectionObservable
        .firstOrError()
        .flatMap { it.readCharacteristic(characteristic.uuid) }
        .observeOn(AndroidSchedulers.mainThread())
        .subscribe({ broadcastCharacteristicUpdate(characteristic.uuid, it) }, { logError(it) })
        .let { disposable.add(it) }
}
```

KUVA 35. Apufunktio readCharacteristic, joka kutsuu kirjaston funktiota readCharacteristic.

Data, jonka readCharacteristic tuottaa, on tyypiltään ByteArray, eli Peripheral-laitteelta luettu raakadata pitää parsia ihmiselle luettavaan muotoon, jotta se on helposti ymmärrettävissä ja hyödynnettävissä. Datan parsimisessa on tärkeää huomioida, että parsittava ByteArray sisältää tavuja GattCharacteristicin sisältämien attribuuttien mukaisesti, eli palautettu arvo saattaa sisältää enemmän kuin yhden tavun. Lisäksi BLE Peripheral-laitteet saattavat järjestään tavunsa myös big-endian-järjestyksellä, vaikka yleisemmin käytetään little-endian-tavujärjestystä. Nämä seikat tulee huomioida, sillä tavujen oikea parsimisjärjestys on datan eheyden kannalta elinehto. (Ong 2020.)

Datan kirjoittaminen Peripheral-laitteelle ei ole aivan yhtä suoraviivainen prosessi kuin datan lukeminen laitteelta. Keskeinen asia GattCharacteristicille kirjoittamisessa on se, että Peripheral-laite saattaa tukea yhtä tai kahta eri kirjoitustyyppiä. Pääasiassa datan kirjoituksessa käytetään WRITE_TYPE_DEFAULT-tyyppiä, sillä yleensä tieto siitä onko kirjoitusoperaatio onnistunut, on merkityksellisempää kuin suurempi datansiirtonopeus. Kirjoitustyypit ovat:

- WRITE_TYPE_DEFAULT, joka on kirjoituspyyntö. Tämä tyyppi edellyttää, että Peripheral-laite vastaa Android-laitteelle joka tapauksessa, eli joko se suorittaa kirjoitustoiminnon ja ilmoittaa operaation onnistumisesta tai kirjoitustoiminto epäonnistuu ja Peripheral-laite ilmoittaa Android-laitteelle epäonnistuneesta kirjoitusoperaatiosta.

- WRITE_TYPE_NO_RESPONSE on kirjoituskomento, joka ei edellytä Peripheral-laitteelta vastausta kummassakaan tapauksessa. (Ong 2020.)

Datan kirjoituksessa on tärkeää ottaa huomioon sallittu kirjoitettavan datan koko. Mikäli Peripheral-laitteelle yritetään kirjoittaa kerralla enemmän tavuja kuin mitä sen ATT Maximum Transmission Unitiksi (ATT MTU) on määritelty, kaatuu operaatio virheeseen GATT_INVALID_ATTRIBUTE_LENGTH. (Ong 2020.)

Kirjoitustyypin lisäksi kirjoitusoperaatio edellyttää, että kirjoitettava data on parsittu ByteArrayksi (Welie 2019). Näin ollen datan kirjoitusoperaatiossa tulee huomioida myös se, mihin tavuun tai tavuihin ByteArrayssa kirjoitusoperaatio kohdistuu. Toisin sanoen, mihinkä käsiteltävänä olevan GattCharacteristicin attribuuttiin kirjoitusoperaatio kohdistuu. Kuvassa 37 kirjoitettava String-tyyppinen arvo muunnetaan ensin funktiolla stringToByteArray ByteArrayksi, mikäli arvon tavukoko ei ylitä attribuutin arvolle sallittua tavukokoa. Myöhemmin kuvan esittämässä koodissa tämä ByteArray-muotoinen arvo asetetaan kyseistä attribuuttia vastaavalle kohdalle käsiteltävänä olevan GattCharacteristicin ByteArray-muotoiseen muuttujaan. Tämä tapahtuu kutsumalla funktiota replaceDataSlice. Tämä päivitetty versio GattCharacteristicin ByteArray-ilmentymästä kirjoitetaan Peripheral-laitteelle käsiteltävänä olevan GattCharacteristicin arvoksi writeCharacteristic-funktiolla.

```
/**
 * Writes a String value to a parameter on the device. Before calling this it's recommended
 * you check that param.string == true. If the parameter is not writable this function
 * does nothing. After writing is done successfully calls ACTION_PARAM_WRITE_SUCCESS.
 *
 * @param param
 * @param value Length of string must be lower or equal than length of the param
 */
fun writeParameter(param: ExampleGattParameter, value: String) {
    val characteristic = examplePeripheral?.spec?.getCharacteristicWithParam(param)
    val btChar = getBtCharacteristic(characteristic)

    // Converts value to byte array
    val byteArray = stringToByteArray(value, param) ?: return

    if (characteristic !== null && btChar !== null) {
        if (characteristic.type.contains(strWrite)) {

            btChar.value = replaceDataSlice(param, characteristic, btChar.value, byteArray)
            connectionObservable
                .firstOrError()
                .flatMap { it: RxBleConnection
                    it.writeCharacteristic(characteristic.uuid, btChar.value)
                }
                .observeOn(AndroidSchedulers.mainThread())
                .subscribe({ it: ByteArray!
                    broadcastParameterUpdate(
                        ACTION_PARAM_WRITE_SUCCESS,
                        characteristic.uuid,
                        btChar.value,
                        param
                    )
                },
                    { logError(it) })
                .let { disposable.add(it) }
        } else {
            return
        }
    }
}
```

KUVA 36. Apufunktio writeParameter, joka kutsuu kirjaston funktiota writeCharacteristic.

4.2.7 Ilmoitusten tai indikaatioiden tilaaminen

Siinä missä Peripheral-laitteelle voidaan suorittaa luku- ja kirjoitusoperaatioita Android-laitteen suunnasta, voidaan siltä myös tilata ilmoituksia (Notifications) tai indikaatioita (Indications) Android-laitteelle. Mikäli Peripheral-laitteelta tilataan ilmoituksia tai indikaatioita, alkaa Peripheral-laite ilmoittaa Android-laitteelle aina, kun sen sisältämä data on päivittynyt. Ilmoituksen lisäksi se lähettää päivitetyn

datan Android-laitteelle. Ilmoitusten ja indikaatioiden välinen ero on se, että ensimmäinen ei vaadi Android-laitteelta vahvistusta datan vastaanotosta, toisin kuin jälkimmäinen. Näin ollen indikaatiot, jotka vaativat Android-laitteelta vahvistuksen ovat ilmoituksia hitaampia. (Ong 2020.)

Ilmoituksia tai indikaatioita tilataan GattCharacteristic-kohtaisesti, eikä niitä ole siis välttämätöntä tilata ollenkaan. GattCharacteristic ei myöskään välttämättä tue ilmoituksia tai indikaatioita, jolloin niiden tilaaminen ei luonnollisesti onnistu. (Ong 2020.) Lisäksi Welie (2019) huomauttaa, että ilmoituksia ja indikaatioita ei voi tilata määräänsä enempää ja että tuo määrä riippuu Android-laitteen käyttöjärjestelmäversiosta. Mitä vanhempi versio, sitä vähemmän ilmoituksia ja indikaatioita Peripheral-laitteelta voi tilata. Viisitoista tilausta on tilausmäärän kattona useimmille laitteille. (Welie 2019.)

GattCharacteristicit, jotka tukevat ilmoitusten ja indikaatioiden tilausta, omaavat Client Characteristic Configuration Descriptorin (CCCD). Ilmoitukset ja indikaatiot merkitään CCCD:hen joko tilatuiksi tai ei-tilatuiksi. CCCD:n arvoksi asetetaan joko ENABLE_INDICATION_VALUE, ENABLE_NOTIFICATION_VALUE tai DISABLE_NOTIFICATION_VALUE ja tämä arvo kirjoitetaan Peripheral-laitteelle. Viimeistä arvovaihtoehtoa käytetään kaikissa tapauksissa, joissa Peripheral-laitteelta ei haluta tilata päivityksiä, eli evätessä ilmoitukset, indikaatiot tai molemmat. Kuvassa 38 esitetään kuinka ilmoitusten ja indikaatioiden tilausta hoidetaan yleensä funktioiden avulla. (Ong 2020.)

```
fun enableNotifications(characteristic: BluetoothGattCharacteristic) {
    val cccdUuid = UUID.fromString(CCC_DESCRIPTOR_UUID)
    val payload = when {
        characteristic.isIndicatable() -> BluetoothGattDescriptor.ENABLE_INDICATION_VALUE
        characteristic.isNotifiable() -> BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE
        else -> {
            Log.e("ConnectionManager", "${characteristic.uuid} doesn't support
notifications/indications")
            return
        }
    }

    characteristic.getDescriptor(cccdUuid)?.let { cccDescriptor ->
        if (bluetoothGatt?.setCharacteristicNotification(characteristic, true) == false) {
            Log.e("ConnectionManager", "setCharacteristicNotification failed for ${characteristic.uuid}")
            return
        }
        writeDescriptor(cccDescriptor, payload)
    } ?: Log.e("ConnectionManager", "${characteristic.uuid} doesn't contain the CCC descriptor!")
}

fun disableNotifications(characteristic: BluetoothGattCharacteristic) {
    if (!characteristic.isNotifiable() && !characteristic.isIndicatable()) {
        Log.e("ConnectionManager", "${characteristic.uuid} doesn't support indications/notifications")
        return
    }

    val cccdUuid = UUID.fromString(CCC_DESCRIPTOR_UUID)
    characteristic.getDescriptor(cccdUuid)?.let { cccDescriptor ->
        if (bluetoothGatt?.setCharacteristicNotification(characteristic, false) == false) {
            Log.e("ConnectionManager", "setCharacteristicNotification failed for ${characteristic.uuid}")
            return
        }
        writeDescriptor(cccDescriptor, BluetoothGattDescriptor.DISABLE_NOTIFICATION_VALUE)
    } ?: Log.e("ConnectionManager", "${characteristic.uuid} doesn't contain the CCC descriptor!")
}
```

KUVA 37. Ilmoitusten ja indikaatioiden tilaaminen (Ong 2020, muokattu).

Ilmoitusten tai indikaatioiden tilaamisen jälkeen koodissa pitää vielä muistaa käsitellä niiden mukana tuleva päivitetty data. Data tulee ByteArray-muodossa ja se kannattaa heti vastaanoton jälkeen kopioida ByteArray-muuttujaan. Mikäli Android-laite vastaanottaa uuden ilmoituksen tai indikaation sillä välin, kun se vielä käsittelee edellisen päivityksen tuomaa dataa, saattaa se ehtiä ylikirjoittaa edellisen päivityksen tuoreemmalla. Tämä säikeistykseen liittyvä ongelma on vältettävissä sillä, että saapuva data otetaan heti sen saavuttua kopiona talteen käsittelyä varten. (Welie 2019.)

4.2.8 Pysyvän yhteyden muodostaminen BLE-laitteen kanssa

Pariutuminen (pairing) ja pysyvän yhteyden muodostaminen (bonding) sekoitetaan herkästi toisiinsa, tai niiden kuvitellaan tarkoittavan samaa toimintoa. Kuluttajanäkökulmasta asialla ei ole niin suurta merkitystä, mutta kehittäjän on tärkeä ymmärtää, että kyseessä on kaksi eri toimintoa, kuten luvussa 2.2.2 Protokollat kerrottiin. Pariutuminen on väliaikaisilla avaimilla suojattu yhteys, joka

lakkaa olemasta, kun yhteys laitteiden välillä katkeaa. Pariutuminen ja siinä vaihdetut väliaikaiset salatut avaimet mahdollistavat myös pysyvän yhteyden muodostuksen ja pitkäikäisten salattujen avaimien vaihdon. Pysyvän yhteyden muodostaminen on näistä kahdesta toiminnosta se, joka kehittäjää kiinnostaa, sillä pariutumisesta huolehtii kummankin laitteen puolella BLE-protokollapino. (Ong 2020.)

Suomenkielisessä tekstissä sekaannusta aiheuttavat herkästi myös termit pysyvä yhteys (bond) ja yhteys (connection). Suomenkielisistä termeistään huolimatta nämä tarkoittavat keskenään täysin eri toimintoja BLE-kontekstissa. Koska termit ovat kuitenkin niin lähellä toisiaan, on tässä opinnäytetyössä pyritty tarpeen mukaan lisäämään englanninkielinen termi suluissa indikoimaan kummasta toiminnosta on kyse.

Pysyvää yhteyttä muodostaessaan Android-laite saattaa pyytää käyttäjältä hyväksyntää, PIN-koodia tai salasanaa tai -lausetta. Tämän jälkeen Android-laite tunnistaa toisen laitteen salaisten avainten avulla, ja yhteys muodostuu käyttäjän näkökulmasta automaattisesti. Pysyvän yhteyden suurin hyöty on se, että yhteys on salattu ja laitteiden toisilleen välittämä data on siten suojattu urkinnalta. (Welie 2019.)

Ong (2020) listaa kolme tapaa, miten Android-laite ja Peripheral-laite voivat muodostaa pysyvän yhteyden:

1. Android-laitteen aloitteesta: Android-laitteet yrittävät itsenäisesti muodostaa pysyvän yhteyden Peripheral-laitteen kanssa tilanteessa, jossa Peripheral-laite antaa Insufficient Authentication -virheen, kun Android-laite yrittää tehdä luvattoman ATT-pyynnön.
2. Android-kehittäjän koodaamasta aloitteesta: Pysyvän yhteyden muodostusprosessin voi käynnistää kutsumalla Androidin BluetoothDevice-luokan metodia createBond.
3. Peripheral-laitteen aloitteesta: Peripheral-laite voidaan laitteen koodaajan toimesta laittaa pyytämään pysyvän yhteyden muodostusta, minkä seurauksena Android-laite yrittää yhdistyä Peripheral-laitteen kanssa pysyvästi. (Ong 2020.)

Welie (2019) listaa samat kolme tapaa yhteyden muodostukselle, mutta lisäksi hän painottaa, että ensisijainen tapa muodostaa pysyvä yhteys laitteiden välillä pitäisi olla aina Android-laitteen itsenäisesti suorittama yhteyden muodostus, joko Insufficient Authentication -virheen seurauksena tai vastauksena Peripheral-laitteen pyyntöön. Hän toteaa, ettei createBond-metodin kutsumiselle ole pääsääntöisesti mitään tarvetta. Welie mainitsee myös, että iOS-käyttöjärjestelmän puolella ei ole olemassa Androidin createBond-metodin kaltaista vaihtoehtoa, jonka avulla kehittäjä voisi koodin kautta tehdä aloitteen pysyvän yhteyden muodostamiseksi. Tähän vedoten hän epäilee, etteivät sellaiset Peripheral-laitteet, joiden kohdalla kehittäjän tarvitsee erikseen kutsua createBond-metodia, ole iOS-yhteensopivia.

Myös Ong (2020) pitää Androidin itsenäisesti suorittamaa yhteyden muodostusta yllättävän luotettavana tapana hoitaa asia, ja tapana, jolla se on parasta hoitaa. Lisäksi hän yhtälailla huomauttaa, ettei iOS-käyttöjärjestelmän puolella ole edes mahdollista käynnistää pysyvän yhteyden muodostusta koodista käsin.

Sellaisissa tapauksissa, joissa Android-laite ei itsenäisesti onnistu muodostamaan pysyvää yhteyttä, createBond-metodi on kuitenkin avuksi. Esimerkiksi jotkin Xiaomin ja Samsungin puhelimista ovat osoittautuneet ongelmallisiksi itsenäisen pysyvän yhteyden muodostuksen suhteen. (Ong 2020.)

Samaan aikaan, kun laitteet yrittävät muodostaa pysyvää yhteyttä, ei niiden välillä saa olla meneillään muita prosesseja. Toisin sanoen, pysyvän yhteyden muodostuksen aikana Peripheral-laitteelta ei saa lukea dataa eikä sinne saa kirjoittaa dataa. Laitteet eivät saa edetä edes Discover Services -vaiheeseen, jossa Android-laite yrittää kartoittaa Peripheral-laitteen tiedostorakennetta. Pysyvän yhteyden muodostuksen aikana ajetut muut prosessit saattavat laukaista erilaisia virheitä tai jopa katkaista laitteiden välisen yhteyden (connection). (Welie 2019.)

Pysyvän yhteyden muodostuksen etenemisen seuranta onnistuu BroadcastReceiverin avulla (kuva 39). BondState, eli pysyvän yhteyden tila, voi olla BOND_NONE, BOND_BONDING tai BOND_BONDED. BroadcastReceiver

rekisteröidään seuraamaan aietta (Intent) ACTION_BOND_STATE_CHANGED. Kun BLE-laite broadcastaa, eli lähettää edellä mainitun aikeen, tulee sen vastaanottajan selvittää aikeen sisältämästä EXTRA_DEVICE-datasta, broadcastaavan, eli lähettävän, laitteen MAC-osoite. Broadcast on luonteeltaan multicast-lähetys, joten BroadcastReceiver vastaanottaa muitakin broadcasteja, kuin mitä sen on tarkoitus seurata. Vastaanottajan on siksi selvitettävä lähettäjän osoite. (Ong 2020.)

BroadcastReceiverin vastaanottama aie sisältää dataa, jota kutsutaan extroiksi. ACTION_BOND_STATE_CHANGED-aikeen extrat ovat:

- EXTRA_DEVICE, datatyypiltään Parcelable
- EXTRA_BOND_STATE, datatyypiltään Int
- EXTRA_PREVIOUS_BOND_STATE, datatyypiltään Int. (Ong 2020.)

```
fun listenToBondStateChanges(context: Context) {
    context.applicationContext.registerReceiver(
        broadcastReceiver,
        IntentFilter(BluetoothDevice.ACTION_BOND_STATE_CHANGED)
    )
}

private val broadcastReceiver = object : BroadcastReceiver() {
    override fun onReceive(context: Context, intent: Intent) {
        with(intent) {
            if (action == BluetoothDevice.ACTION_BOND_STATE_CHANGED) {
                val device = getParcelableExtra<BluetoothDevice>(BluetoothDevice.EXTRA_DEVICE)
                val previousBondState = getIntExtra(BluetoothDevice.EXTRA_PREVIOUS_BOND_STATE, -1)
                val bondState = getIntExtra(BluetoothDevice.EXTRA_BOND_STATE, -1)
                val bondTransition = "${previousBondState.toBondStateDescription()} to " +
                    bondState.toBondStateDescription()
                Log.w("Bond state change", "${device?.address} bond state changed | $bondTransition")
            }
        }
    }

    private fun Int.toBondStateDescription() = when(this) {
        BluetoothDevice.BOND_BONDED -> "BONDED"
        BluetoothDevice.BOND_BONDING -> "BONDING"
        BluetoothDevice.BOND_NONE -> "NOT BONDED"
        else -> "ERROR: $this"
    }
}
```

KUVA 38. BroadcastReceiver ja pysyvän yhteyden tilan seuraaminen (Ong 2020, muokattu).

Mikäli pysyvän yhteyden tilan muutoksista ei ole tarve olla selvillä, vaan olennaista on vain selvittää, mikä tämän hetkinen tila on, Androidin BluetoothDevice tarjoaa metodin getBondState, joka palauttaa jonkin kolmesta mahdollisesta tilamoodista, eli BOND_NONE, BOND_BONDING tai BOND_BONDED.

Pysyvän yhteyden katkaisu ei onnistu koodista käsin, eli sen toteutukseen ei ole tarjolla BluetoothDevicella julkista metodia. Ainoa tapa millä, kehittäjä voi auttaa käyttäjää katkaisemaan pysyvän yhteyden, on ohjeistamalla tätä tekemään se puhelimen tai muun laitteen asetuksista käsin. (Ong 2020.)

4.2.9 Taustalle jäävän yhteyden ylläpito, kun laitteiden välillä ei ole pysyvää yhteyttä

Pariutuminen ja pysyvä yhteys takaavat kumpikin omalla tavallaan suojatun yhteyden laitteiden välille, kuten luvussa 2.2.2 Protokollat on kuvattu. Mikäli laitteiden välistä kommunikaatiota ei ole tarpeen salata, ei laitteiden ole pakko muodostaa pysyvää yhteyttä tai pariutua keskenään voidakseen kommunikoida toistensa kanssa. Esimerkiksi tämän opinnäytetyön taustalla olevassa projektissa, asiakas halusi, ettei pariutuminen tapahdu automaattisesti, joten asiakkaan toimittamien Peripheral-laitteiden koodissa oli huomioitu, että pariutuminen suoritetaan vasta sitä pyytävän käyttäjäsyötteen perusteella.

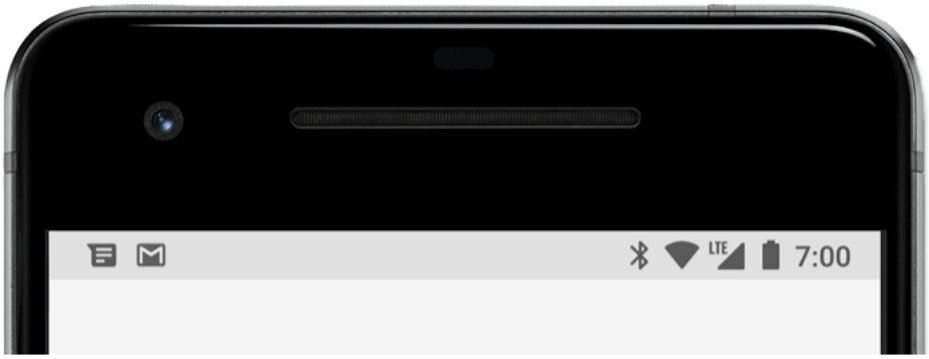
BLE-yhteyden (connection) omistus, ja täten hallinta, riippuu BLE-laitteiden pysyvän yhteyden tilasta (bondState). Kun laitteet eivät ole keskenään pysyvässä yhteydessä, BLE-yhteyden omistaa sovellus. Toisaalta tilanteessa, jossa pysyvä yhteys laitteiden väliltä löytyy, yhteyden omistaa Android-käyttöjärjestelmä. Käytännössä tämä tarkoittaa sitä, että sovelluksissa, joissa Android-laite ja Peripheral-laite eivät ole muodostaneet keskenään pysyvää yhteyttä, mutta ovat yhteydessä toisiinsa, tämä yhteys katkeaa, mikäli Android-käyttöjärjestelmä sulkee sovelluksen resurssinhallintasyistä tai käyttäjä sulkee sovelluksen pyyhkäisemällä sen pois ruudulta. (Ong 2020.)

Esimerkiksi musiikin suoratoistosovellus voi menettää yhteyden BLE-kuulokkeisiin, jos sovellus pyyhkäistään pois Android-laitteen pois ruudulta. Toisaalta liikunnan seurantaan laadittu sovellus saattaa ylläpitää yhteyttä BLE-laitteisiin ruudulta poispyyhkäisyn jälkeenkin.

Ong (2020) kertoo, että suoraviivaisin tapa yrittää pitää sovellus ja BLE-yhteys hengissä silloinkin, kun pysyvää yhteyttä laitteiden välillä ei ole, on käyttää

Androidin Foreground Serviceä (etualan palvelu). Tässä tapauksessa puhutaan Androidin komponentista Service, eikä BLE-datarakenteeseen kuuluvasta datan organisointitasosta. Ong tähdentää, ettei Foreground Servicejen käyttökään tuo Android BLE-yhteydelle 100 % toimintavarmuutta. Vaikka Foreground Servicen käyttö pienentää riskiä, että Android-käyttöjärjestelmä sulkee sovelluksen resurssihallintasyistä, ei se kuitenkaan täysin poista sitä. (Ong 2020.)

Android Developers (2022) kirjoittaa Foreground Serviceistä, että ne jatkavat toimintaansa, vaikkei käyttäjä olisi vuorovaikutuksessa sovelluksen kanssa. Käyttäjän tulee kuitenkin olla tietoinen siitä, että sovellus jatkaa yhä toimintaansa, vaikkei se ole aktiivisesti ruudulla, joten Foreground Servicet joutuvat näyttämään Android-ilmoitusta (Notification) Android-laitteen ruudulla niin kauan kuin ne ovat ajossa (kuva 40). Tätä ilmoitusta ei voi poistaa muuten kuin siten, että käyttäjä yksiselitteisesti sulkee sovelluksen. (Android Developers 2022.)



KUVA 39. Android-ilmoitukset puhelimen tilapalkissa (Android Developers 2022).

Kun käyttäjä pyyhkäisee pois sellaisen sovelluksen, joka käyttää Foreground Serviceä, kaikki sovelluksen käyttäjärajapinnan elementit, eli käytännössä kokonaisuudessaan Androidin Activity-pino, pyyhkiytyvät pois. Käyttäjän näkökulmasta sovellus saattaa siis näyttää suljetulta. Tällainen sovellus kuitenkin jatkaa toimintaansa taustalla. Mikäli sovellus sisältää BLE-yhteyden hallintaa ja mikäli hallintaan liittyvä logiikka on rakennettu muualle kuin pois pyyhkiytyvään Activity-pinoon, säilyy BLE-yhteys siinä missä muukin sovelluksen taustalle jäävä toiminta. (Ong 2020.)

4.2.10 Kolmannen osapuolen BLE-kirjastot

Android BLE -projekti on täysin mahdollista rakentaa ilman kolmannen osapuolen kirjastoa. Toisaalta, kuten tämän opinnäytetyön taustalla olevassa projektissa tehtiin, on Android-laitteille rakennettavien BLE-hallintasovellusten kokoonpanossa mahdollista hyödyntää kolmannen osapuolen BLE-kirjastoja. Näiden tarkoitus on helpottaa Android BLE APIen hallintaa, sillä kuten Ong (2020) ja Welie (2019) toteavat, Android BLE APIen virallinen dokumentaatio on merkittävältä osin vajavaista, välillä peräti harhaanjohtavaa ja epävirallinen dokumentaatio on pirstaleina internetissä.

Kolmannen osapuolen kirjaston käyttämisessä on myös riskinsä. Ei ole taattua, että kolmannen osapuolen kirjastoa ylläpidetään siten, että se päivittyy Android-päivitysten kanssa samassa tahdissa. Tämä voi johtaa tilanteisiin, joissa kirjasto tai sen jotkin ominaisuudet menevät rikki, kun Android saa uuden päivityksen eivätkä niiden toiminnot enää kooditasolla kohtaa. Muita mahdollisia ongelmia kolmannen osapuolen kirjastojen käytössä ovat lisensointi ja IP-asiat. Nämä koskettavat lähinnä yrityksien tekemiä sovelluksia. (Ong 2020.)

Ong (2020) listaa joitakin PunchThrough-yrityksen hyväksi toteamia ja luotettavia avoimen lähdekoodin kirjastoja:

- **RxAndroidBle**, jonka pohjalla on RxJava. Tämän kirjaston eduksi voidaan mainita se, että kirjasto sisältää paljon rajapintoja ja on täten helposti testattavissa.
- **Nordic Android BLE Library**, joka toimii erityisen hienosti yhteen Nordic Semiconductor -laiteohjelmistojen kanssa. Tämän kirjaston käyttö poistaa virallisen Android BLE SDK:n kompleksisuutta ja vaaroja.
- **BleGattCoroutines**, jonka pohjalla on Kotlin coroutines. Tätä suositellaan, jos Kotlinin coroutines on kehittäjälle entuudestaan tuttu tai Kotlin coroutnesia käytetään projektissa laajasti. (Ong 2020.)

4.3 Software Development Kit eli SDK

IBM Cloud Education (2021) määrittelee SDK:n seuraavasti: SDK on sovelluskehitystyökalujen setti, joka sisältää rakennuspalikoita, debuggereita, eli virheentarkastajia, ja usein myös kehyksen (Framework) tai koodikirjastoryhmän, jolla hoidetaan käyttöjärjestelmälle tyypillisiä rutiineita. SDK sisältää useimmiten myös vähintään yhden APIn, sillä ilman sovellusrajapintaa sovellukset eivät voi kommunikoida keskenään. (IBM Cloud Education 2021.)

Kehittäjät käyttävät SDK:ita, sillä ne nopeuttavat sovellusten kirjoittamista tarjoamalla tiettyjä toimintoja helposti käytettävässä paketissa. SDK:iden käyttö tukee myös sovellusten standardoitumista. Tyypillisesti SDK ladataan ja asennetaan kehitysalustalle, minkä jälkeen sen tarjoamat työkalut ovat kehitettävän sovelluksen käytettävissä. (IBM Cloud Education 2021.)

4.3.1 Android-kirjasto

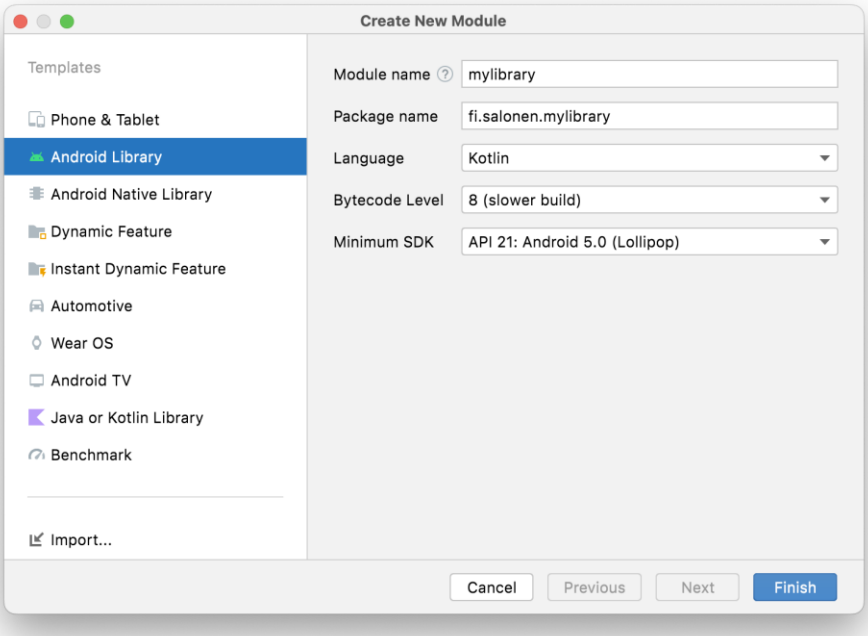
Android-kirjasto on rakenteellisesti sama asia kuin Android-sovellusmoduuli. Android-kirjastoa ei kuitenkaan rakenneta itsenäisten sovellusten käyttämään APK-formaattiin vaan siitä rakennetaan Android Archive -tiedosto (AAR), jota voidaan käyttää riippuvuutena muissa Android-sovelluksissa. Android-kirjasto on käytännöllinen esimerkiksi silloin, kun tehdään useita sovelluksia, jotka jakavat samat keskeiset toiminnot. (Android Developers 2022.)

Android AAR-tiedosto eroaa Android JAR-tiedostosta siten, että toisin kuin JAR-tiedosto, AAR-tiedosto voi sisältää resursseja kuten manifest-tiedoston, joka antaa paketoida mukaan sovelluksen layout-tiedostot ja drawable-tiedostot sovelluksen luokkien ja metodien lisäksi (Geeks for Geeks 2021). AAR-tiedostot eroavat myös siten JAR-tiedostoista, että ne voivat sisältää C- ja C++-kirjastoja (Android Developers 2022).

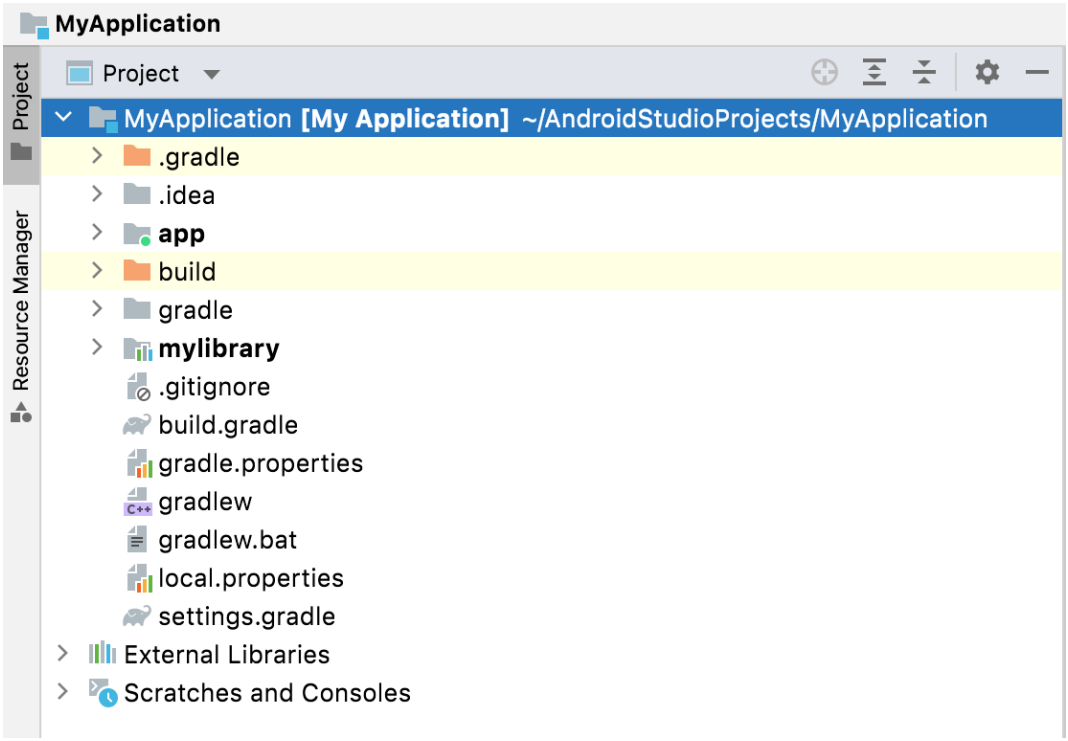
Opinnäytetyön taustalla olevassa projektissa asiakkaan tilaama SDK toteutettiin Android-kirjastona. Asiakkaalle toimitettiin kirjastotiedoston lisäksi dokumentaatio sen toiminnasta ja siitä, miten se asennetaan Android-sovellusprojektiin riippuvuudeksi.

4.3.2 Android-kirjaston luominen Android Studiossa

Android-kirjaston luominen Android Studiossa edellyttää sitä, että Android Studioon on luotu Android-projekti. Tämän jälkeen luodaan uusi kirjastomoduuli seuraavasti: **File > New > New module**. Tämän jälkeen aukeaa ikkuna (kuva 41), jonka vaihtoehtovalikosta valitaan **Android Library**. Kirjastomoduulin voi nimetä haluamallaan tavalla ja valita haluamansa alimman SDK-version. Tämän jälkeen moduulin luonti suoritetaan klikkaamalla **Finish-nappia.** Luotu kirjastomoduuli ilmestyy projektinäkymään omana moduulikansionaan (kuva 42).



KUVA 40. Android Studion ikkuna Create New Module (Luo uusi moduuli).



KUVA 41. Projektinäkymä, jossa on mukana kirjastomoduuli ’mylibrary’.

Toisaalta olemassa olevasta sovelluksesta voi konvertoida kirjastomoduulin muokkaamalla moduulitason build.gradle tiedostoa. Tiedostosta poistetaan rivi, jolla määritellään applicationId (kuva 43), sillä ainoastaan Android-sovellusmoduulit voivat määritellä tämän arvon. (Android Developers 2022.)

```
defaultConfig {
    applicationId "fi.salonen.myapplication"
```

KUVA 42. Moduulitason build.gradle-tiedoston rivi ’applicationId’.

Seuraavaksi samasta tiedostosta muutetaan tiedoston alusta kuvassa 44 esitetty rivi ’id’ kuvassa 45 esitettyyn muotoon. Näiden muutosten jälkeen Android Studio pyytää synkronoimaan projektin, mikä on syytä tehdä. Projektista voi nyt rakentaa AAR-tiedoston valitsemalla ylävalikosta **Build > Build APK**. Koska build.gradle-tiedostoon tehtiin edellä kuvatut muutokset, rakentaa Android Studio APK-tiedoston sijasta AAR-tiedoston. (Android Developers 2022.)

```
plugins {
    id 'com.android.application'
```

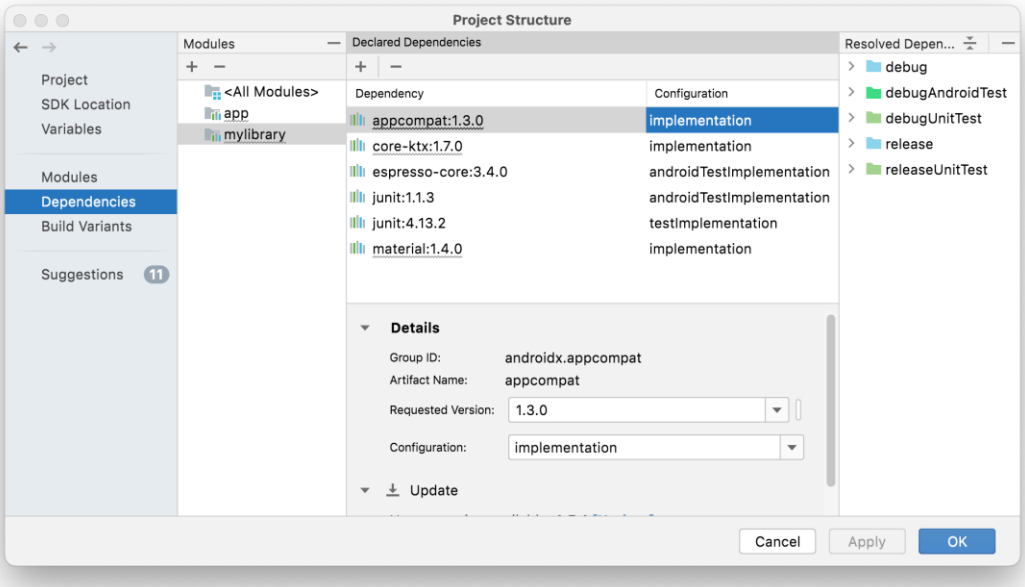
KUVA 43. Moduulitason build.gradle-tiedoston rivi 'id'.

```
plugins {
    id 'com.android.library'
```

KUVA 44. Moduulitason build.gradle-tiedoston rivi 'id'.

4.3.3 Android-kirjaston käyttäminen projektissa

Android-kirjaston käyttäminen samassa projektissa, johon se on luotu, tapahtuu siten, että kirjastomoduuli lisätään projektin riippuvuudeksi. Valitsemalla **File > Project Structure > Dependencies** aukeaa kuvassa 46 esitetty ikkuna. (Android Developers 2022.)



KUVA 45. Android Studion Project Structure -ikkuna.

Seuraavaksi valitaan moduuleista se, jolle riippuvuutta ollaan luomassa ja klikataan **Declared Dependencies -kentän plus-merkkiä**. Aukeavasta valikosta valitaan vaihtoehto **Module Dependency** ja seuraavaksi avautuvassa dialogi-

ikkunassa valitaan haluttu kirjastomoduuli. **OK-nappia** klikkaamalla Android Studio tekee tarvittavan muutoksen moduulitason build.gradle-tiedostoon (kuva 47). (Android Studio 2022.)

```
dependencies {

    implementation project(path: ':mylibrary')
```

KUVA 46. Moduulitason build.gradle-tiedoston muutos, joka lisää halutun kirjaston saman projektin riippuvuudeksi.

Kirjastomoduulin käyttäminen muissa kuin siinä Android-projektissa, johon se luotiin, edellyttää sitä, että kirjastomoduulista paketoidaan AAR-tiedosto. Tämä tapahtuu helpoiten terminaalista käsin. Terminaalissa siirrytään projektin juureen. Projektin juuresta löytyvät tiedostot gradlew ja gradlew.bat. Näistä ensimmäinen on Mac- ja Linux-yhteensopiva ja jälkimmäinen vuorostaan Windows-yhteensopiva Gradle Wrapper komentorivityökalu. Käyttöjärjestelmästä riippuen ajetaan terminaalissa komento:

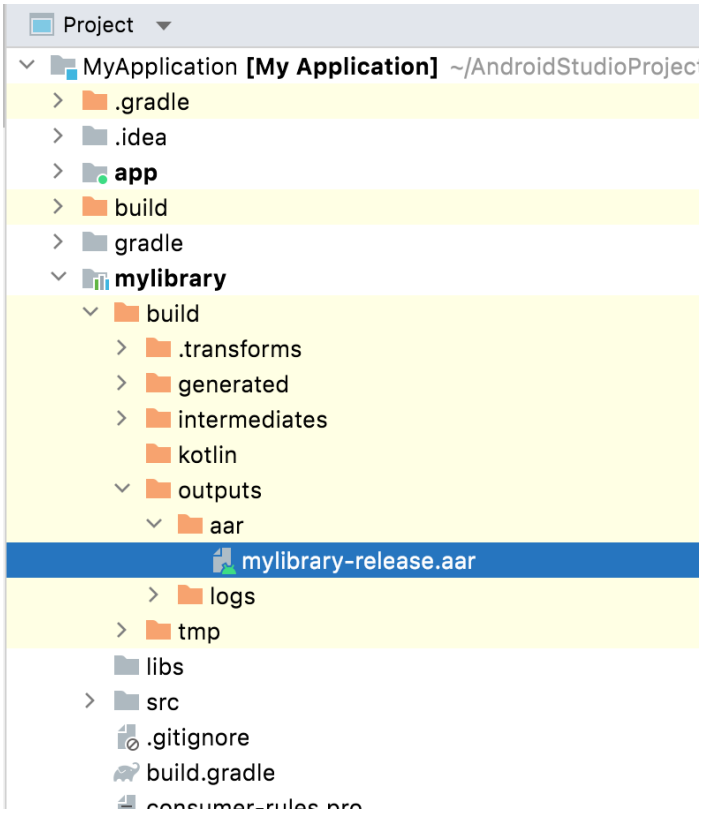
Mac / Linux:
`./gradlew assembleRelease`

Windows:
`gradlew.bat assembleRelease` (Windows)

Yllä esitetyn komennon ajaminen tuottaa AAR-tiedoston kirjastomoduulin sisälle `mylibrary/build/outputs/aar/`-kansioon (kuva 48). Tämän tiedoston voi siirtää tai kopioida haluamaansa projektiin. Projektissa, jossa kirjastoa halutaan käyttää, tallennetaan AAR-tiedosto `project/app/libs/`-kansioon. Tämän lisäksi projektin moduulitason build.gradle-tiedoston riippuvuuksiin (dependencies) tulee lisätä rivi:

```
implementation fileTree(include: ['*.aar'], dir: 'libs')
```

Aina kun build.gradle tiedostoihin tehdään muutoksia, projekti tulee synkronoida, jotta päivitykset tulevat voimaan. Projektin synkronoinnin jälkeen kirjasto on projektin käytettävissä.



KUVA 47. Uudelleen käytettävän AAR-tiedoston sijainti kirjastomoduulin lähdekoodin sisällä.

5 POHDINTA

Opinnäytetyössä oli tarkoitus selvittää mitä Bluetooth ja Bluetooth Low Energy tarkoittavat ja toisaalta myös, miten Android-laitteet ja Bluetooth Low Energy -laitteet kommunikoivat keskenään. Työn tarkoituksena oli laatia dokumentaatio aiheesta toimeksiantajayritykselle ja tavoitteena syventää toimeksiantajayrityksen sovelluskehittäjien osaamista työn kattamilla osa-alueilla.

Androidin Bluetooth Low Energy -sovellusrajapinnat ovat kompleksisia ja paikoin varsin heikosti dokumentoituja, joten vankka ymmärrys Bluetoothin ja Bluetooth Low Energyn toiminnasta tukee niiden parissa työskentelyä. Niinpä tässä opinnäytetyössä ei keskitytty ainoastaan ruotimaan sitä, miten Android BLE APIt toimivat, vaan panostettiin myös selvitykseen siitä, miten niin sanotun Bluetooth-raudan puoli toimii.

Bluetooth Classic ja BLE-laitteiden väliset erot osoittautuivat melko merkittäviksi. Mielenkiintoinen käänne Bluetoothin kehityksessä on ollut se, että siirrettävän datan koon loputtoman kasvattamisen sijaan Bluetooth onkin ”kesken kaiken” vaihtanut suuntaa ja tähdännyt IoT-markkinoille pienellä virrankulutuksella ja suuremmalla kantosäteellä. Laitekantojen erojen ymmärtäminen tukee keskeistä Bluetooth-osaamista.

Tässä opinnäytetyössä perehdyttiin BLE APIen toimintaan, sillä niiden käyttöä helpottavien kirjastojen dokumentaatiot osoittautuivat paikoitellen vajavaisiksi. Näin ollen BLE APIen toiminnan ymmärtäminen helpottaa myös kolmannen osapuolen kirjastojen käyttöä.

Opinnäytetyön tuloksena syntyi liitteestä 1 löytyvä kevyt koonti opinnäytetyön sisällöstä toimeksiantajan käsikirjaa varten. Koska tietoa opinnäytetyön aiheeseen liittyen kertyi merkittävästi enemmän ja syvemmin kuin on järkevää litteroida nopeasti omaksuttavaksi tarkoitettuun käsikirjamerkintään, opinnäytetyön tekijä pitää toimeksiantajayrityksen henkilöstölle koulutuksen aiheeseen liittyen.

LÄHTEET

Android Developers. 27.10.2021. Bluetooth Low Energy. Verkkosivu. Viitattu 23.10.2022. https://developer.android.com/guide/topics/connectivity/bluetooth/ble-overview

Android Developers. 27.10.2021. Connect to a GATT server. Verkkosivu. Viitattu 30.10.2022. https://developer.android.com/guide/topics/connectivity/bluetooth/connect-gatt-server

Android Developers. 31.8.2022. Services overview. Verkkosivu. Viitattu 3.11.2022. https://developer.android.com/guide/components/services

Android Developers. 24.10.2022. Create an Android library. Verkkosivu. Viitattu 3.11.2022. https://developer.android.com/studio/projects/android-library

Android Developers. 24.10.2022. Notifications overview. Verkkosivu. Viitattu 3.11.2022. https://developer.android.com/develop/ui/views/notifications

Afaneh, M. 2018. Intro to Bluetooth Low Energy. Novel Bits. E-kirja. Viitattu 15.10.2022. Vaatii käyttöoikeuden. https://novelbits.s3.us-east-2.amazonaws.com/Website/Lead+Magnets/Intro+to+Bluetooth+Low+Energy+v1.1.pdf

Bluetooth. 2022. Learn about Bluetooth, Recent Enhancement Overview. Verkkosivu. Viitattu 9.10.2022. https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/

Bluetooth. 2022. Learn about Bluetooth, Mesh Networking. Verkkosivu. Viitattu 9.10.2022. https://www.bluetooth.com/learn-about-bluetooth/recent-enhancements/mesh/

Bluetooth. 2022. Technology Overview. Verkkosivu. Viitattu 5.10.2022. https://www.bluetooth.com/learn-about-bluetooth/tech-overview/

Bluetooth SIG Proprietary. 6.12.2016. Bluetooth Core Specification v.5.0. Bluetooth. E-kirja. Viitattu 16.10.2022. https://www.bluetooth.com/specifications/specs/core-specification-5/

Bui, H. 10.12.2015. Bluetooth Smart and the Nordic Softdevices – Part 1 GAP Advertising. Nordic semiconductor. Verkkosivu. Viitattu 22.10.2022. https://devzone.nordicsemi.com/guides/short-range-guides/b/bluetooth-low-energy/posts/bluetooth-smart-and-the-nordics-softdevices-part-1

Callaham, J. 13.8.2022. The history of Android: The evolution of the biggest mobile OS in the world. Android Authority. Verkkosivu. Viitattu 25.10.2022. https://www.androidauthority.com/history-android-os-name-789433/

Geeks For Geeks. 17.3.2021. Different Ways to Create aar File in Android Studio. Verkkosivu. Viitattu 3.11.2022. https://www.geeksforgeeks.org/different-ways-to-create-aar-file-in-android-studio/

Get Connected Blog. 26.5.2021. The Difference Between Classic Bluetooth and Bluetooth Low Energy. Nordic semiconductor. Verkkosivu. Viitattu 9.10.2022. https://blog.nordicsemi.com/getconnected/the-difference-between-classic-bluetooth-and-bluetooth-low-energy

GSMArena Team. 29.12.2010. Bluetooth 3.0 and Wireless N Speed test: Something in the air. GSMArena. Verkkosivu. Viitattu 9.10.2022. https://www.gsmarena.com/wireless_n_bluetooth_3_speed_test-review-551p4.php

Hanna, K. & Burke, J. 2021. Definition frequency-hopping spread spectrum. TechTarget. Verkkosivu. Viitattu 10.10.2022. https://www.techtarget.com/searchnetworking/definition/frequency-hopping-spread-spectrum

Heinzman, A. 15.2.2022. What Is Multipoint Bluetooth and How Does It Work? Review Geek. Verkkosivu. Viitattu 9.10.2022. https://www.reviewgeek.com/109925/what-is-multipoint-bluetooth-and-how-does-it-work/

Hlapisi, N. 18.7.2022. Bluetooth Low Energy Stack: Understanding the Layers. Novel Bits. Verkkosivu. Viitattu 11.10.2022. https://novelbits.io/bluetooth-low-energy-protocol-stack-layers/

Hlapisi, N. 1.8.2022. Bluetooth ATT and GATT explained (Connection Oriented Communication). Novel Bits. Verkkosivu. Viitattu 23.10.2022. https://novelbits.io/bluetooth-le-att-gatt-explained-connection-oriented-communication/

Lindevall, K. 2022. Bluetooth-kommunikaatio Android-sovelluksella peltokoneen kanssa. Tampereen Ammattikorkeakoulu. Opinnäytetyö. Viitattu 16.10.2022. https://www.theseus.fi/bitstream/handle/10024/746637/Lindevall_Kalle.pdf?sequence=2&isAllowed=y

IBM Cloud Education. 13.7.2021. SDK vs. API: What’s the Difference? IBM. Verkkosivu. Viitattu 3.11.2022. https://www.ibm.com/cloud/blog/sdk-vs-api

Jimblom. n.d. Bluetooth Basics. SparkFun. Verkkosivu. Viitattu 16.10.2022. https://learn.sparkfun.com/tutorials/bluetooth-basics/bluetooth-profiles

LM Technologies. 2020. AT Command Reference. Verkkosivu. Viitattu 11.10.2022. https://wiki.lm-technologies.com/at-command-reference/

Marcel, J. 2.3.2018. From Handsets to Hands Free, The Birth of Bluetooth Connectivity. Bluetooth. Verkkosivu. Viitattu 5.10.2022. https://www.bluetooth.com/blog/from-handsets-to-hands-free/

Marks, T. 30.8.2022. How Secure is Bluetooth? A Full Guide to Bluetooth Safety. VPNOverview.com. Verkkosivu. Viitattu 16.10.2022. https://vpnoverview.com/privacy/devices/bluetooth/

Microchip. 2021. Bluetooth® Low Energy GAP Roles. Verkkosivu. Viittattu 22.10.2022. https://microchipdeveloper.com/wireless:ble-gap-roles

Moumita. 22.5.2020. The Bluetooth Protocol Architecture. Tutorials Point. Verkkosivu. Viitattu 11.10.2022. https://www.tutorialspoint.com/the-bluetooth-protocol-architecture

Moumita. 22.5.2020. The Bluetooth Protocol Stack. Tutorials Point. Verkkosivu. Viitattu 11.10.2022. https://www.tutorialspoint.com/the-bluetooth-protocol-stack

Nguen, A. 18.4.2018. Bluetooth 1.0 vs 2.0 vs 3.0 vs 4.0 vs 5.0 – How They Compare | Symmetry Blog. Symmetry Electronics. Verkkosivu. Viitattu 9.10.2022. https://www.symmetryelectronics.com/blog/bluetooth-1-0-vs-2-0-vs-3-0-vs-4-0-vs-5-0-how-they-compare-symmetry-blog/

Ong, C. 10.9.2019. 4 Tips to Make Android BLE Actually Work. Punch Through. Verkkosivu. Viitattu 29.10.2022. https://punchthrough.com/android-ble-development-tips/

Ong, C. 15.5.2020. The Ultimate Guide to Android Bluetooth Low Energy. Punch Through. Verkkosivu. Viitattu 29.10.2022. https://punchthrough.com/android-ble-guide/

Peter. 16.5.2021. Flashback: a brief history of Bluetooth. GSMArena. Verkkosivu. Viitattu 5.10.2022. https://www.gsmarena.com/flashback_a_brief_history_of_bluetooth-news-49119.php

Polidea. 29.8.2016. RxAndroidBLE – Your most powerfull tool for Bluetooth Low Energy coding! Medium. Verkkosivu. Viitattu 30.10.2022. https://medium.com/@polidea_15491/rxandroidble-your-most-powerful-tool-for-bluetooth-low-energy-coding-c38eaf6e6d76

Pothitos, A. 2.8.2017. The History of Bluetooth. Mobile Industry Review. Verkkosivu. Viitattu 5.10.2022. https://www.mobileindustryreview.com/2017/08/the-history-of-bluetooth.html

RF Wireless World. 2012. Piconet vs Scatternet-Difference between Piconet and Scatternet. RF Wireless World. Verkkosivu. Viitattu 16.10.2022. https://www.rfwireless-world.com/Terminology/difference-between-piconet-and-scatternet-in-bluetooth.html

Scientific American. 5.11.2007. How Does Bluetooth Work? Verkkosivu. Viitattu 5.10.2022. https://www.scientificamerican.com/article/experts-how-does-bluetooth-work/

Sony Electronics Inc. 26.5.2022. Bluetooth Wireless Technology Profiles and Descriptions. Verkkosivu. Viitattu 16.10.2022.

https://www.sony.com/electronics/support/speakers-wireless-speakers/lf-s50g/articles/00007501

Staab, W. & Armstrong, S. 13.1.2014. Bluetooth 101 – Part VI – Bluetooth Architecture. Hearing Health & Technology Matters. Verkkosivu. Viitattu 16.10.2022. https://hearinghealthmatters.org/waynesworld/2014/bluetooth-101-part-vi/

Tawil, Y. 29.9.2018. Bluetooth Low Energy (BLE) 101 Tutorial: Intensive Introduction. Atadiat. Verkkosivu. Viitattu 9.10.2022. https://atadiat.com/en/e-bluetooth-low-energy-ble-101-tutorial-intensive-introduction/

Tawil, Y. 22.6.2017. Why it's called Bluetooth Stack. Atadiat. Verkkosivu. Viitattu 16.10.2022. https://atadiat.com/en/e-why-its-called-bluetooth-stack/

Texas Instruments. 2016. Overview. Verkkosivu. Viitattu 22.10.2022. https://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.35.00.33/exports/docs/ble5stack/ble_user_guide/html/ble-stack/overview.html

Texas Instruments. 2016. Generic Access Profile (GAP). Verkkosivu. Viitattu 22.10.2022. https://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.35.00.33/exports/docs/ble5stack/ble_user_guide/html/ble-stack/gap.html

Texas Instruments. 2016. Generic Attribute Profile (GATT). Verkkosivu. Viitattu 23.10.2022. https://software-dl.ti.com/lprf/simplelink_cc2640r2_sdk/1.35.00.33/exports/docs/ble5stack/ble_user_guide/html/ble-stack/gatt.html

Townsend, K. 20.3.2014. Introduction to Bluetooth Low Energy. Adafruit. Verkkosivu. Viitattu 22.10.2022. https://learn.adafruit.com/introduction-to-bluetooth-low-energy?view=all

Townsend, K. 22.10.2022. GAP. Adafruit. Verkkosivu. Viitattu 22.10.2022. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gap

Townsend, K. 23.10.2022. GATT. Adafruit. Verkkosivu. Viitattu 23.10.2022. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

Triggs, R. & Wankhede, C. 17.6.2022. A little history of Bluetooth. Android Authority. Verkkosivu. Viitattu 5.10.2022. https://www.androidauthority.com/history-bluetooth-explained-846345/

Welie, M. 23.3.2019. Making Android BLE work – part 1. Medium. Verkkosivu. Viitattu 29.10.2022. https://medium.com/@martijn.van.welie/making-android-ble-work-part-1-a736dcd53b02

Welie, M. 6.4.2019. Making Android BLE work – part 2. Medium. Verkkosivu. Viitattu 29.10.2022. https://medium.com/@martijn.van.welie/making-android-ble-work-part-2-47a3cdaade07

Welie, M.15.4.2019. Making Android BLE work – part 3. Medium. Verkkosivu. Viitattu 2.11.2022. https://medium.com/@martijn.van.welie/making-android-ble-work-part-3-117d3a8aee23

Welie, M.16.4.2019. Making Android BLE work – part 4. Medium. Verkkosivu. Viitattu 2.11.2022. https://medium.com/@martijn.van.welie/making-android-ble-work-part-4-72a0b85cb442

Woodford, C. 4.6.2021. How does Bluetooth work? Explain that Stuff. Verkkosivu. Viitattu 5.10.2022. https://www.explainthatstuff.com/howbluetooth-works.html

Woolley, M. 9.9.2021. Bluetooth® Core Specification Version 5.0 Feature Enhancements. Bluetooth. E-kirja. Viitattu 5.10.2022. https://www.bluetooth.com/bluetooth-resources/bluetooth-core-specification-version-5-3-feature-enhancements/?utm_campaign=specification&utm_source=internal&utm_medium=blog&utm_content=new-core-specification-v5.3-feature-enhancements

Yang, J., Poellabauer, C., Mitra P. & Neubecker C. 25.9.2019. Beyond Beaconing: Emerging Applications and Challenges of BLE. Research Gate. E-kirja. Viitattu 16.10.2022. https://www.researchgate.net/publication/336084528_Beyond_Beaconing_Emerging_Applications_and_Challenges_of_BLE

LIITTEET

Liite 1. Toimeksiantajan käsikirjaa varten laadittu dokumentaatio

Bluetooth-versiot

Bluetooth jaetaan versioidensa perusteella Bluetooth Classic -laitteisiin ja Bluetooth Low Energy -laitteisiin (BLE). Bluetooth-versiot 1.0–3.0 kuuluvat Classic -kantaan ja tähän mennessä julkaistut versiot 4.0:sta eteenpäin ovat BLE-kantaa. Bluetooth Classic ja BLE eivät ole yhteensopivia, vaan Classic-laitteet voivat keskustella vain Classic-laitteiden kanssa ja BLE-laitteet vain toisten BLE-laitteiden kanssa. Poikkeuksena tähän sääntöön ovat niin sanotut Dual Mode -laitteet kuten monet älypuhelimet, jotka tukevat kumpaakin laitekantaa.

Bluetooth Classic

Bluetooth on lyhyen kantaman langattoman yhteyden teknologia, joka hyödyntää yhteyksissään samoja radiotaajuuksia kuin esimerkiksi Wi-Fi-reititin. Se operoi luvasta vapaiden radiolähettimien ISM-kaistalla, joka on jaettu 79 kanavaan ja keskitetty 2.4 gigahertsiin.

Bluetooth Low Energy

Bluetooth Low Energyä varten ISM-kaistalle tehtiin uusi jako, jonka myötä BLE-laitteet käyttävät 40 kanavaa, joista kolme on Advertisement-kanavia. Advertisement on BLE:n esittelemä ominaisuus, jonka avulla BLE-laitteet lähettävät (Broadcast) ”mainosdataa”, eli kertovat vastaanottavalle laitteelle läsnäolostaan.

Frequency hopping

Bluetoothin toiminta nojaa FHSS-teknologiaan (frequency-hopping spread spectrum). Tämän teknologian avulla Bluetooth jakaa taajuuskaistan pienempiin kanaviin ja hyppii tiheästi, pseudosatunnaisella, muuttuvalla kaavalla kanavasta toiseen datasignaaleja siirtäessään. Sekä lähettävä että vastaanottava laite tuntevat muuttuvan kaavan, ja kanava vaihtuu aina yhden datasignaalin siirron jälkeen.

Topologiat

Bluetooth Classicilla on vain yksi topologia. Se toimii laitteiden välisellä Point-to-Point-yhteydellä, joka muodostaa piconetin, langattoman ad hoc -verkon, jossa voi olla saman aikaisesti yksi Master-laite ja korkeimmillaan seitsemän Slave-laitetta. Piconetissa olevat laitteet voivat kuulua samaan aikaan muihinkin Piconeteihin, joilloin muodostuvasta verkosta käytetään nimeä Scatternet.

BLE:n myötä Point-to-Point-yhteyden rinnalle tuli kaksi uutta topologiaa, Broadcast ja Mesh. Broadcast-verkkoa käyttävät sellaiset BLE-laitteet, joiden tarkoitus on ainoastaan lähettää pieniä datamääriä lukuisille laitteille. Mesh-yhteyden avulla BLE-laitteilla voidaan rakentaa laajan skaalan verkkoja. Mesh-verkkojen kautta voidaan ohjata, monitoroida ja automatisoida järjestelmiä, joissa on satoja tai jopa tuhansia laitteita, jotka keskustelevat keskenään.

Profiilit

Bluetooth-profiilit määrittelevät mihin Bluetooth-laitetta käytetään. Bluetooth-laitteen profiili tai profiilit kertovat siis, mitä käyttötarkoitusta varten laite on. Esimerkiksi autossa käytettävä hands free -kuulokesetti käyttää headset-profiilia (HSP) ja Nintendo Wii -ohjain käyttää vuorostaan human interface device -profiilia (HID). Jotta Bluetooth-laitteet voivat toimia yhdessä, tulee niiden molempien tukea samaa profiilia.

Bluetooth-protokollat

Bluetooth-arkkitehtuuri ei noudata standardoitua OSI- tai TCP/IP-mallia, vaan sillä on oma itsenäinen malli protokollapinoilleen. Bluetooth poikkeaa perinteisistä verkkoarkkitehtuureista myös siten, ettei sitä tukevien laitteiden tarvitse toteuttaa kaikkia pinon protokollia, vaan Bluetoothia hyödyntävä sovellus määrittelee sen, mitä niistä se käyttää. Bluetooth Classicin protokollapino eroaa jonkin verran BLE:n protokollapinosta ja Dual Mode -laitteet toteuttavat kumpaakin näistä protokollapinoista. BLE-laitteiden kommunikoinnin kannalta, sovellusten kirjoittamista ajatellen, kiinnostavimmat kerrokset ovat BLE-pinon ylimmät, eli sovellusta lähinnä olevat, kerrokset.

BLE-protokollapinon ylimmät kerrokset

BLE-protokollapinon ylimmät kerrokset, Generic Access Profile (GAP), Attribute Protocol (ATT) ja Generic Access Attribute (GATT), muodostavat protokollat, joiden mukaan BLE-laiteet keskustelevat keskenään. Generic Access Profile -taso vastaa yhteyden muodostamiseen liittyvistä toiminnoista. GAP-taso hallinnoi saavutettavuuteen liittyviä proseduureja ja moodeja kuten laitteen havaitseminen (Device Discovery), linkin muodostaminen (Link Establishment), linkin katkaisu (Link Termination), turvallisuusominaisuuksien käynnistäminen (Initiation of Security Features) ja laitteen konfiguraatio (Device Configuration).

ATT-tason protokollat määrittelevät, mitä dataa palvelinlaite luovuttaa asiakaslaitteelle. Palvelinlaite on siis BLE-laite, joka hyväksyy vertaiseltaan laitteelta sisään tulevat komennot, lähettää vastauksia, ilmoituksia (Notifications) ja viittauksia (Indications). Asiakaslaite toimii vuorostaan palvelinlaitteen käyttäjärajapintana. Asiakaslaitteella annetaan komentoja ja pyyntöjä (Requests) palvelinlaitteelle, luetaan palvelinlaitteen lähettämään dataa ja hyväksytään palvelinlaitteelta tulevat ilmoitukset ja viittaukset. ATT-tasolla palvelinlaitteen säilömä tai keräämä, ja välittämä data on jäsennelty attribuuttitietokantaan.

Attribuutti on minkä tahansa tyyppistä palvelinlaitteen asiakaslaitteelle välittämää dataa. Attribuutit rakentuvat aina seuraavasti:

- attribuuttityyppi (Universally Unique Identifier or UUID)
    - 16-bittinen (SIGiltä adoptoitu attribuutti) tai 128-bittinen (toimittajakohtainen attribuutti) numero
- attribuuttikahva (Handle)
    - 16-bittinen arvo, jonka palvelin asettaa jokaiselle attribuutille
    - ”osoite”, jonka perusteella asiakas voi kohdentaa toimensa nimetylle attribuutille
    - arvo on muuttumaton koko laitteiden välisen yhteyden ajan
- attribuutin luvat (Permissions)
    - määrittelevät saako attribuutille kirjoittaa ja saako sen sisältämän arvon lukea sekä voiko sille tehdä ilmoituksia tai viittauksia
    - määrittelevät turvallisuustasot kullekin yllä mainitulle toiminnalle
- attribuutin arvo (Value) attribuutin sisältämä muuttuja-arvo.

GATT-taso on palveluviitekehys (Service Framework), joka määrittelee aliproseduurit, joilla ATT-tason toimintoja säädellään. BLE-laitteet siirtyvät GATT-tasolle, kun ne ovat muodostaneet yhteyden keskenään. GATT-tason protokollat vastaavat laitteiden välisestä kommunikaatiosta ja datansiirrosta.

GATT-taso voidaan nähdä tiedostorakenteena, jossa kaiken kehyksenä toimii Profile. Profile sisältää Servicen tai useampia. Service sisältää Characteristicin tai useampia ja Characteristic sisältää Valueita ja näiden Descriptoreita.
Profile ei ole BLE-laitteella itsestään olemassa, vaan se valitaan ja asetetaan joko SIGin ennalta määrittelemistä profiileista (ks. kohta Profiilit). Kun palvelinlaitteelle on määritelty jokin profiili, tulee sen toteuttaa tämän profiilin määrittelemät Servicet ja Characteristicsit. Jotkin profiilissa määritellyistä Serviceistä ja Chracteristicseista voivat olla vapaavalintaisia, mutta toiset niistä on pakko toteuttaa, mikäli palvelinlaite ilmoittaa olevansa yhdenmukainen tietyn profiilin kanssa.

GATT-Service on pikemminkin datan organisointiin liittyvä yksikkö kuin tallennusyksikkö. Aivan kuten tietokoneella olla useita kansioita, voi BLE-laite myöskin sisältää enemmän kuin yhden Servicen. Samaan analogiaan nojaten, kansio sisältää siihen liittyviä tiedostoja ja Service vastaavasti siihen liittyviä Chracteristicseja. Jokainen Service saa tunnistusta varten oman UUID:n. Tämä UUID voi olla joko 16-bittinen virallisesti adoptoidulle BLE Servicelle tai 128-bittinen, jolloin se on BLE-laitteen valmistajan oma kustomoitu Service-UUID.

GATT-Characteristic on perus tallennusyksikkö, johon sovelluksen dataa kirjoitetaan, tai josta sitä luetaan. Se vastaa tiedostoa tietokoneella. Jokaisen Characteristicin määrittelee aina Value (arvo) ja mahdollisesti Descriptor (kuvaus). Value sisältää sovellukseen tallennetun datayksikön. Vapaavalintaisesti käytettävä Descriptor pitää sisällään lisäinformaatiota siihen liittyen. Kukin Characteristic saa tunnisteekseen UUID:n. Tämä UUID on joko 16-bittinen virallisesti adoptoitu, SIGin määrittelemä, BLE-Chracteristic tai 128-bittinen BLE-laitteen valmistajan määrittelemä Characteristic-UUID.

Tietoturva

Bluetoothin mahdolliset tietoturvahaavoittuvuudet voidaan jakaa kolmeen kategoriaan:

1. passiivinen salakuuntelu (passive eavesdropping)
2. man-in-the-middle-hyökkäykset (MITM)
3. identiteetin jäljitys (identity tracking).

BLE:n osalta passiivista salakuuntelua vastaan on varauduttu Link Layerin tasolla tapahtuvalla AES-CCM-salauksella. MITM-hyökkäyksiä varten BLE suojautuu paritusvaiheessa. Identiteetin jäljitystä vastaan BLE taistelee muuttamalla laitteen osoitetta määräajoin.

Android BLE API

Android tarjoaa sisäänrakennettuna tuen BLE Central-laitteen roolille ja useampia rajapintoja, joiden avulla sovellukset voivat löytää BLE Peripheral-laitteita, tehdä Service-kyselyjä ja siirtää dataa.

KOLMANNEN OSAPUOLEN KIRJASTOT

Android-laitteille rakennettavien BLE-hallintasovellusten kokoonpanossa mahdollista hyödyntää kolmannen osapuolen BLE-kirjastoja. Näiden tarkoitus on helpottaa Android BLE APIen hallintaa, sillä Android BLE APIen virallinen dokumentaatio on merkittävältä osin vajavaista, välillä peräti harhaanjohtavaa ja epävirallinen dokumentaatio on pirstaleina internetissä.

PROJEKTIN PERUSTAMINEN

Android-BLE-projektia perustettaessa on syytä asettaa alimmaksi APIn kohdetasoksi (minimum target API) 21, jonka vastine Android versioissa on 5.0, sillä sen tuoman päivityksen myötä Google paransi Androidin BLE-toiminnallisuutta merkittävästi kehittäjäystävällisempään suuntaan. Androidille esiteltiin ohjelmointirajapinnat BluetoothLeScanner ja ScanFilter ja APIen toiminnasta tuli muutenkin monin tavoin vakaampaa. Suositeltu ohjelmointikieli on Kotlin.

TARVITTAVAT LUVAT

Projektin käynnistysvaiheessa on syytä kiinnittää huomiota tarvittavien lupien käsittelyyn (Permission Handling). Android-projekteissa luvilla tarkoitetaan niitä käyttölupia, joita Android-laite pyytää käyttäjältä tämän avatessa sovelluksen. Luvat määritellään projektin AndroidManifest.xml-tiedostossa.

Tarvittavat luvat tapauksessa, jossa alin API kohdetaso on 21:

- BLUETOOTH; antaa sovellukselle luvan muodostaa yhteyden Bluetooth-laitteen kanssa.
- BLUETOOTH_ADMIN; antaa sovellukselle luvan skannata Bluetooth-laitteita pysyvän yhteyden muodostusta varten.
- ACCESS_FINE_LOCATION; vaaditaan Android 6.0 -versiosta alkaen, jotta skannaustulokset tulevat näkyviin. BLE-skannaus voi tahattomasti paljastaa käyttäjän sijainnin kehittäjille, jotka skannaavat tiettyjä BLE-majakoita, joten tämä tarkan sijainnin lupa on käyttäjänsuojan nimissä vaadittu.

Edellä mainituista luvista ainoastaan sijaintiin liittyvä lupa lasketaan Android-termein vaaralliseksi luvaksi. Tämä tarkoittaa sitä, lupa on yksiselitteisesti ja selkeästi pyydettävä käyttäjältä. Muut tarvittavat luvat ovat sellaisia, ettei niihin tarvita käyttäjältä erillistä suostumusta, vaan ne hyväksytään sovelluksen asennuksen yhteydessä.

Huomion arvoista on, että käyttäjärajapinnassa tulee tehdä Android-laitteen Bluetoothin päällä olo -tarkistus sekä käsitellä ACCESS_FINE_LOCATION-lupaan liittyvä käyttäjän antama vastaus. Android BLE -sovellus ei voi toimia ilman päällä olevaa Bluetoothia ja skannaus ei tuota vastauksia ilman lupaa tarkkaan sijaintiin.

Mikäli alin APIn kohdetaso on 31 tai enemmän, tulee AndroidManifest.xml-tiedostossa määritellä seuraavat luvat:

- BLUETOOTH_SCAN
- BLUETOOTH_ADVERTISE
- BLUETOOTH_CONNECT
- ACCESS_FINE_LOCATION

SKANNAUS

Ennen kuin BLE-laitteeseen voi muodostaa yhteyden, tulee kyseinen laite löytää BLE-skannauksen avulla. Skannaus suoritetaan BluetoothLeScannerilla, joka on BluetoothAdaterin objekti. BluetoothAdapter on vuorostaan Android-laitteen Bluetooth-raudan ilmentymä. Se tarjoaa informaatiota Bluetooth-raudan on/off -tilasta, mahdollistaa pysyvän yhteyden BLE-laiteyhteyksille tehdyt kyselyt ja BLE-skannauksen aloituksen. Skannausta käynnistettäessä on hyvä tarkistaa, onko skannaus jo käynnissä ja pysäyttää se ja käynnistää uudelleen, mikäli tämä on tilanne.

Skannaus aloitetaan kutsumalla BluetoothLeScannerin startScan-metodia. Skannaukselle voidaan määritellä filtereitä ScanFilterin avulla, mutta tämä ei ole välttämätöntä. Filtteröinnillä voidaan rajata skannauksessa löytyviä laitetyyppejä. Skannauksen filtteröinti voidaan suorittaa joko BLE Services-UUID:lla, merkkijonolla, joka sisältää hyväksyttävän laitteen nimen tai hyväksyttävän laitteen MAC-osoitteen. Filtterit tulee määritellä ennen skannauksen aloittamista.

Tulosten filtteröinnin lisäksi skannaukselle voidaan määritellä myös asetuksia, joiden mukaan skannaus suoritetaan. Asetukset määritellään ScanSettings-luokan avulla. Skannauksen asetukset on määriteltävä ennen kuin skannaus käynnistetään.

Skannaus pitää saada katkaistua koska tahansa myös suoralla komennolla. Esimerkiksi käyttäjälle on syytä tarjota mahdollisuus katkaista aloitettu skannaus halutessaan, joten on syytä luoda funktio tällaista komentoa varten. Skannaus pitää lopettaa myös siinä tilanteessa, kun käyttäjä päättää muodostaa yhteyden Peripheral-laitteen kanssa. Skannauksen pysäyttäminen onnistuu BluetoothLeScanner.stopScan-funktiota kutsumalla.

YHTEYDEN MUODOSTUS (CONNECTION)

Käytännössä Peripheral-laitteisiin on mahdollista muodostaa yhteys kahdella eri tavalla, joko yhteys muodostetaan automaattisesti tai se muodostetaan manuaalisesti. Manuaalinen yhteyden muodostus tarkoittaa sitä, että käyttäjä valitsee skannauksen tuloksista laitteen, johon haluaa muodostaa yhteyden Android-laitteellaan. Automaattinen yhteyden muodostus tapahtuu vuorostaan itsestään, ilman käyttäjän syötettä. Automaattista yhteyden muodostusta käytetään ainoastaan tapauksissa, jolloin tiedetään tarkasti Peripheral-laitteen lähettämän Advertisement-datan perusteella, että kyseessä on haluttu Peripheral-laite.

BLE-laitteilla automaattinen yhteyden muodostus on riippuvainen seuraavista asioista:

- Löytyykö ennakkoon määritelty Peripheral-laite skannauksella tavoitetuista Advertisement-datapaketeista?
- Onko Android-laitteen ja Peripheral-laitteen välille muodostettu pysyvä yhteys (bonding)?
- Löytyykö Peripheral-laite Android-laitteen cachesta?

Yhteyden muodostus tapahtuu siten, että Android-laite muodostaa connectGatt-metodin avulla yhteyden GATT-serveriin, eli Peripheral-laitteeseen. Metodi connectGatt palauttaa BluetoothGatt-luokan instanssin ja callback-funktion Peripheral-laitteelta. Saadun instanssin ja callback-funktion avulla Android-laite voi suorittaa Client-laitteen toimintoja.

DISCOVERING SERVICES

Jotta Android-laitteella voidaan suorittaa operaatioita Peripheral-laitteelle, tulee Android-laitteen tuntea Peripheral-laitteen datarakenne. Gatt-Servicejen löytäminen on siis käytännössä pakollinen seuraava askel sen jälkeen, kun BLE-laitteiden välinen yhteys on onnistuneesti muodostettu.

LUKU- JA KIRJOITUSOPERAATIOT

Android BLE API:en luku- ja kirjoitusoperaatiot ovat asynkronisia, mikä tarkoittaa sitä, että lähettyyn komentoon tulee vastaus välittömästi, mutta data, jota komennolla luetaan tai kirjoitetaan, liikkuu hitaammin callback-funktion välityksellä. Lisäksi Androidin BluetoothGatt-lähdekoodiin on tehty toiminnallisuus, joka estää ajamasta useampaa kuin yhtä asynkronista operaatiota kerrallaan. Koska asynkronisia operaatioita voi ajaa vain yhden kerrallaan, on ennen uuden operaation käynnistämistä odotettava aina, että aiemmin aloitetun operaation callback-funktio on ensin ajettu loppuun.

Asynkronisuuden lisäksi BluetoothGattCharacteristicsien luku- ja kirjoitusoperaatioissa tulee ottaa huomioon, että nämä operaatiot onnistuvat ainoastaan, jos ne on määritelty kirjoitettaviksi ja luettaviksi. Mikäli luku- tai kirjoitusoperaatio ei ole sallittu, se kaatuu virheeseen GATT_READ_NOT_PERMITED- tai GATT_WRITE_NOT_PERMITED-virheeseen. BluetoothGattCharacterisitic sisältää getProperties-metodin, jonka avulla voi selvittää salliiko kyseinen characteristic luku- tai kirjoitus- tai luku- ja kirjoitusoperaatioita.

Data, jonka readCharacteristic tuotaa on tyypiltään ByteArray, eli Peripheral-laitteelta luettu raakadata pitää vielä parsia ihmiselle luettavaan muotoon, jotta se on helposti ymmärrettävissä ja hyödynnettävissä. BLE Peripheral-laitteet saattavat järjestään tavunsa myös big-endian-järjestyksellä, vaikka yleisemmin käytetään little-endian-tavujärjestystä. Nämä seikat huomioiden, tavujen oikea parsimisjärjestys on datan eheyden kannalta elinehto.

Keskeinen asia GattCharacteristicille kirjoittamisessa on se, että Peripheral-laite saattaa tukea yhtä tai kahta eri kirjoitustyyppiä. Nämä kirjoitustyypit ovat:

- WRITE_TYPE_DEFAULT, joka on kirjoituspyyntö. Tämä tyyppi edellyttää, että Peripheral-laite vastaa Android-laitteelle joka tapauksessa, eli joko se suorittaa kirjoitustoiminnon ja ilmoittaa operaation onnistumisesta Android-laitteelle tai kirjoitustoiminto epäonnistuu ja Peripheral-laite ilmoittaa Android-laitteelle epäonnistuneesta kirjoitusoperaatiosta.

- WRITE_TYPE_NO_RESPONSE on kirjoituskomento, joka ei edellytä Peripheral-laitteelta vastausta kummassakaan tapauksessa. (Ong 2020.)

Datan kirjoituksessa on tärkeää ottaa huomioon sallittu kirjoitettavan datan koko. Mikäli Peripheral-laitteelle yritetään kirjoittaa kerralla enemmän tavuja kuin mitä sen ATT Maximum Transmission Unitiksi (ATT MTU) on määritelty, kaatuu operaatio virheeseen GATT_INVALID_ATTRIBUTE_LENGTH.

Kirjoitustyypin lisäksi kirjoitusoperaatio edellyttää, että kirjoitettava data on parsittu ByteArrayksi. Datan kirjoitusoperaatiossa tulee huomioida myös se, mihin tavuun tai tavuihin ByteArrayssa kirjoitusoperaatio kohdistuu. Toisin sanoen, mihinkä käsiteltävänä olevan GattCharacteristicin attribuuttiin kirjoitusoperaatio kohdistuu.

DESCRIBING NOTIFICATIONS & INDICATIONS

Siinä missä Peripheral-laitteelle voidaan suorittaa luku- ja kirjoitusoperaatioita Android-laitteen suunnasta, voidaan siltä myös tilata ilmoituksia (Notifications) tai indikaatioita (Indications) Android-laitteelle. Mikäli Peripheral-laitteelta tilataan ilmoituksia tai indikaatioita, alkaa Peripheral-laite ilmoittaa Android-laitteelle, mikäli sen sisältämä data on päivittynyt ja lisäksi se lähettää päivitetyn datan Android-laitteelle. Ilmoitusten ja indikaatioiden välinen ero on se, että ensimmäinen ei vaadi Android-laitteelta vahvistusta datan vastaanotosta, toisin kuin jälkimmäinen.

BONDING

Pariutuminen (pairing) ja pysyvän yhteyden muodostaminen (bonding) sekoitetaan herkästi toisiinsa, tai niiden kuvitellaan tarkoittavan samaa toimintoa. Kuluttajanäkökulmasta asialla ei ole niin suurta merkitystä, mutta kehittäjän on tärkeä ymmärtää, että kyseessä on kuitenkin kaksi eri toimintoa. Suomenkielisessä tekstissä sekaannusta aiheuttavat herkästi myös termit pysyvä yhteys (bond) ja yhteys (connection). Suomenkielisistä termeistään huolimatta nämä tarkoittavat keskenään täysin eri toimintoja BLE-kontekstissa.

Pariutuminen ja siinä vaihdetut väliaikaiset salatut avaimet mahdollistavat pysyvän yhteyden muodostuksen ja pitkäikäisten salattujen avaimien vaihdon. Pysyvän yhteyden muodostaminen on näistä kahdesta toiminnosta se, joka kehittäjää kiinnostaa, sillä pariutumisesta huolehtii kummankin laitteen puolella BLE-protokollapino.

Pysyvää yhteyttä muodostaessaan Android-laite saattaa pyytää käyttäjältä hyväksyntää, PIN-koodia tai salasanaa tai -lausetta. Tämän jälkeen Android-laite tunnistaa toisen laitteen salaisten avainten avulla, ja yhteys muodostuu käyttäjän näkökulmasta automaattisesti. Pysyvän yhteyden suurin hyöty on se, että yhteys on salattu ja laitteiden toisilleen välittämä data on siten suojattu urkinnalta.

YHTEYDEN YLLÄPITO TAUSTALLA, KUN PYSYVÄÄ YHTEYTTÄ EI OLE MUODOSTETTU

Paras tapa yrittää pitää sovellus ja BLE-yhteys hengissä silloinkin, kun pysyvää yhteyttä laitteiden välillä ei ole, on käyttää Androidin Foreground Serviceä. Foreground Servicejen käyttökään ei tuo tämän tyyppiselle Android BLE-yhteydelle 100 % toimintavarmuutta. Vaikka Foreground Servicen käyttö pienentää riskiä, että Android-käyttöjärjestelmä sulkee sovelluksen resurssihallintasyistä, ei se kuitenkaan täysin poista sitä.

Android-kirjasto

Android-kirjasto on rakenteellisesti sama asia kuin Android-sovellusmoduuli. Android-kirjastoa ei kuitenkaan rakenneta itsenäisten sovellusten käyttämään APK-formaattiin vaan siitä rakennetaan Android Archive -tiedosto (AAR), jota voidaan käyttää riippuvuutena muissa Android-sovelluksissa.

Android AAR-tiedosto eroaa Android JAR-tiedostosta siten, että toisin kuin JAR-tiedosto, AAR-tiedosto voi sisältää resursseja kuten manifest-tiedoston, joka antaa paketoida mukaan sovelluksen layout-tiedostot ja drawable-tiedostot sovelluksen luokkien ja metodien lisäksi. AAR-tiedostot eroavat myös siten JAR-tiedostoista, että ne voivat sisältää C- ja C++-kirjastoja.

ANDROID-KIRJASTON LUOMINEN

Android-kirjaston luominen Android Studiossa edellyttää sitä, että Android Studioon on luotu Android-projekti. Tämän jälkeen luodaan uusi kirjasto moduuli seuraavasti: **File > New > New module**, minkä seurauksena ruudulle aukeaa ikkuna, jonka vaihtoehtovalikosta valitaan **Android Library**. Kirjastomoduulin voi nimetä haluamallaan tavalla ja valita haluamansa alimman SDK-version. Tämän jälkeen moduulin luonti suoritetaan klikkaamalla **Finish-nappia.** Luotu kirjastomoduuli ilmestyy projektinäkymään omana moduulikansionaan

ANDROID-KIRJASTON KÄYTTÄMINEN SIINÄ PROJEKTISSA, JONKA ALLE SE LUOTIIN

Android-kirjaston käyttäminen samassa projektissa, johon se on luotu, tapahtuu siten, että kirjastomoduuli lisätään projektin riippuvuudeksi. Siirrytään **File > Project Structure > Dependencies**. Seuraavaksi valitaan moduuleista se, jolle riippuvuutta ollaan luomassa ja klikataan **Declared Dependencies -kentän plus-merkkiä**. Aukeavasta valikosta valitaan vaihtoehto **Module Dependency** ja seuraavaksi avautuvassa dialogi-ikkunassa valitaan haluttu kirjastomoduuli. **OK-nappia** klikkaamalla Android Studio tekee tarvittavan muutoksen moduulitason build.gradle-tiedostoon.

ANDROID-KIRJASTON KÄYTTÄMINEN MUISSA PROJEKTEISSA

Kirjastomoduulin käyttäminen muissa kuin siinä Android-projektissa, johon se luotiin, edellyttää sitä, että kirjastomoduulista paketoidaan AAR-tiedosto. Tämä tapahtuu helpoiten terminaalista käsin. Terminaalissa siirrytään projektin juureen. Projektin juuresta löytyvät tiedostot gradlew ja gradlew.bat. Näistä ensimmäinen on Mac- ja Linux-yhteensopiva ja jälkimmäinen vuorostaan Windows-yhteensopiva Gradle Wrapper komentorivityökalu. Käyttöjärjestelmästä riippuen ajetaan terminaalissa komento:

Mac / Linux:

```
./gradlew assembleRelease
```

Windows:

`gradlew.bat assembleRelease` (Windows)

Komennon ajaminen tuottaa AAR-tiedoston kirjastomoduulin `mylibrary/build/outputs/aar/`-kansioon. Tämän tiedoston voi siirtää tai kopioida haluamaansa projektiin. Projektissa, jossa kirjastoa halutaan käyttää, tallennetaan tämä AAR-tiedosto `project/app/libs/`-kansioon. Lisäksi projektin moduulitason build.gradle-tiedoston riippuvuuksiin (dependencies) tulee lisätä rivi:

```
implementation fileTree(include: ['*.aar'], dir: 'libs')
```

Aina kun build.gradle tiedostoihin tehdään muutoksia, projekti tulee synkronoida, jotta päivitykset tulevat voimaan. Projektin synkronoinnin jälkeen kirjasto on projektin käytettävissä.

Liite 2. Android-käyttöjärjestelmäversiot (Callaham 2022)

Version	Code name	Release year	Some features
1.0	-	2008	Android OS, Google Maps, YouTube, pre-Chrome HTML browser w/ Google Search, Android Market
1.5	Cupcake	2009	upload to YouTube, display rotation support for 3rd party keyboard
1.6	Donut	2009	quick search box, quick Camera toggle, Gallery, power control for Wi-Fi, Bluetooth, GPS, etc.
2.0–2.1	Eclair	2009	text-to-speech support, wallpapers, multi-account support, Google Maps navigation
2.2	Froyo	2010	Wi-Fi hotspot, push notifications via Android Cloud to device messaging, flash support
2.3	Gingerbread	2010	UI refresh, NFC support for devices w/ hardware to support it, support for multiple cameras
3.0	Honey Comb	2011	UI for larger displays
4.0	Ice Cream Sandwich	2011	Favorites tray on home screen, biometric device unlocking (photo of user's face), swipe gestures to dismiss notifications and browser tabs
4.1–4.3	Jelly Bean	2012–2013	larger notifications w/ buttons, full Chrome support, external displays and Miracast, improved touch responsiveness, HDR photography support
4.4	KitKat	2013	optimized Android to run 512 MB of RAM
5.0	Lollipop	2014	1st to use Material Design language, UI updates, official support for dual-SIM, HD Voice calls, Device Protection
6.0	Marshmallow	2015	biometric device unlocking (fingerprint), USB-C port, Android Pay, vertically scrolling app drawer
7.0	Nougat	2016	split screen mode, quick switching between apps, JIT compiler for faster apps, Vulkan API for faster 3D rendering

8.0	Oreo	2017	UI updates on Settings menu, native support for picture-in-picture mode, notification channels, new autofill APIs to improve password and other fill data management
9.0	Pie	2018	removal of navigation buttons, on-device machine learning, better battery life, Shush, Slices
10	Q	2019	support for foldable phones, system wide dark mode, new gesture controls, smart reply features for all messaging apps, more control over app-based permissions
11	Red Velvet Cake	2020	Conversations notification category for all chats across apps, record home screen w/ audio, section dedicated to control iOT devices
12	Snow Cone	2021	Material You UI overhaul, scrollable screen captures, App Search, Nearby Share (Wi-Fi), One-handed mode, audio selection access from Media Player
13	Tiramisu	2022	BLE Audio support, updated Now Playing widget, revamped silent mode, Fast Pair (BLE)