Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Gašper Kojek

Primerjava učinkovitosti prenosa podatkov v standardih Bluetooth

magistrsko delo

magistrski program druge stopnje
računalništvo in informatika

Mentor: izr. prof. dr. Patricio Bulić
Somentor: doc. dr. Anton Biasizzo

Ljubljana, 2019

University of Ljubljana
Faculty of Computer and Information Science

Gašper Kojek

Comparing data transmission efficiency in Bluetooth Standards

master's thesis

the 2nd cycle master's study programme
computer and information science

Supervisor: izr. prof. dr. Patricio Bulić
Co-supervisor: doc. dr. Anton Biasizzo

Ljubljana, 2019

Povzetek

Naslov: Primerjava učinkovitosti prenosa podatkov v standardih Bluetooth

Ta magistrska naloga predstavlja opravljeno delo, potrebno za meritve in primerjavo energetske učinkovitosti tehnologije Bluetooth Low Energy (BLE). Predstavljene so primerjave porabe električne energije glede na pretočnost med različnimi verzijami standarda BLE.

Predstavljena je kratka zgodovina tehnologije Bluetooth Low Energy, kot tudi glavne izboljšave in razlike med verzijami specifikacij tehnologije. Meritve smo opravili z enim parom naprav, kateri sta imeli parametre nastavljene glede na različico BLE, na kateri je test temeljil. Porabo energije smo merili z različnimi parametri, pri branju, pisanju, obveščanju in pisanju brez odziva.

Po pričakovanjih so rezultati pokazali, da novejše verzije BLE ponujajo večjo pretočnost, pa tudi dejstvo, da je povprečna poraba energije neodvisna od količine poslanih podatkov. Zanimivo je, da se z vsako novo različico BLE poraba energije na preneseno količino podatkov zmanjša, tudi če se poveča pretočnost. Ugotovili smo tudi, da je energija, potrebna za prenos določene količine podatkov, konstantna pri spreminjanju pretočnosti pri določeni različici specifikacije. Ugotovitve v tej diplomski nalogi dokazujejo da ni razloga, da ne bi prešli na razvoj in uporabo naprav, ki uporabljajo tehnologijo Bluetooth Low Energy 5, saj prinaša večjo pretočnost, večji doseg in boljše sožitje z drugimi brezžičnimi tehnologijami, hkrati pa porabi isto ali manjšo količino energije za prenos podatkov.

Ključne besede

Bluetooth, zmogljivost, pretočnost, energetska učinkovitost, meritve, nizka poraba

Abstract

Title: Comparing data transmission efficiency in Bluetooth Standards

This master thesis presents the work done to measure and compare power efficiency of Bluetooth Low Energy technology versions. More specifically, comparisons between different BLE core specification versions are made with regards to power consumption in relation to throughput.

A brief history of Bluetooth Low Energy is presented, as well as the main improvements and differences between core specification versions. Measurements were performed with a single pair of devices, which had their connection parameters tuned in line with BLE version they were targeting. Power consumption was measured with different parameters, with read, write, notify and write without response BLE operations.

As expected, results revealed that newer BLE versions offer higher throughput, as well as the fact that average power consumption is independent from the amount of data transmitted. Interestingly enough, with each BLE version, power consumption per transmitted amount of data is decreasing, even when the throughput is increased. It was also discovered that total energy needed to transmit some amount of data is constant when changing the throughput in a single core specification version. The findings in this thesis prove that there is no reason not to switch to developing and using devices that use Bluetooth Low Energy 5, as BLE 5 brings higher throughput, extended range and better coexistence while consuming the same or lower amount of energy for transmissions.

Keywords

Bluetooth, performance, throughput, power efficiency, measurements, low energy

Acknowledgments

First and foremost I would like to thank my fiancee Eva for all the love and support she gave me during the writing of this thesis and through the course of my studies.

Furthermore, I would like to thank my family and friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis.

I would like to thank my mentor izr. prof. dr. Patricio Bulić for his help with my thesis. I would also like to thank my co-mentor doc. dr. Antoin Biasizzo from Jožef Stefan Institute for his counselling and guidance with the technical challenges I encountered during my research.

A special thanks goes to Foolography GmbH and its CEO Oliver Perialis for providing me with the hardware needed to conduct the measurements and kind-hearted support.

Gašper Kojek, 2019

Contents

List of used acronyms

acronym	meaning
IoT	Internet of Things
SIG	Special Interest Group
BR/EDR	Basic Rate/Enhanced Data Rate
BLE	Bluetooth Low Energy
HS	High Speed
LR	Long Range
PHY	physical layer
L2CAP	Logical Link Control and Adaptation Protocol
GAP	Generic Access Profile
ATT	Attribute Protocol
GATT	Generic Attribute Profile
MTU	Maximum Transmission Unit
DLE	Data Length Extension
SoC	System on Chip
FEC	Forward Error Correction
UUID	Universally Unique Identifier
CE	Connection Event
CI	Connection Interval
SoC	System-on-Chip
MCU	Microcontroller Unit
DK	Development Kit
SDK	Software Development Kit

kB	kilobyte (1024 bytes)
kbps	kilobits per second (1024 bits per second)
Mbps	mega bits per second (1024 * 1024 bits per second)
GFSK	Gaussian Frequency Shift Keying
FHSS	Frequency-hopping Spread Spectrum

Razširjeni povzetek

V zadnjih letih smo lahko opazili velik napredek v digitalnih tehnologijah, katere prispevajo k vedno večjemu pojmu "internet stvari"(*Internet of Things - IoT*). Velika večina IoT izdelkov uporablja za komunikacijo Bluetooth, tako da je le-ta še vedno v razvoju in se nenehno izboljšuje. Kljub temu nismo zasledili nobene študije, ki bi primerjala porabo energije v primerjavi s pretokom pri različnih parametrih povezave in z različnimi različicami standarda Bluetooth Low Energy. To je še posebej zanimivo v povezavi z novim standardom Bluetooth 5, ki obljublja dvakratno hitrost, štirikraten doseg in osemkrat boljšo zmogljivost oddajanja, ki naj ne bi vplival na porabo moči. Predstavili smo primerjave porabe električne energije glede na pretočnost med različnimi verzijami specifikacije BLE.

I Uvod

Primerjava porabe energije za prenos podatkov z uporabo različnih tehnologij ni nič novega, je pa izjemno zanimivo, da nismo našli nobene take raziskave, ki bi primerjala različne verzije tehnologije Bluetooth. Našli smo nekaj raziskav, ki primerjajo Bluetooth Low Energy v4.0 ali v4.1 z ostalimi brezžičnimi tehnologijami. V članku "How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4"[1], kjer so primerjali BLE v4.0 z ZigBee tehnologijo, so ugotovili da BLE resnično porabi majhno količino energije glede na poslane podatke. V članku "Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and

SimpliciTI"[2] so prav tako uporabljali BLE v4.0, katerega so primerjali z IEEE 802.15.4 in SimpliciTI tehnologijami. Tudi avtorji tega članka so prišli do podobnih zaključkov: Bluetooth Low Energy ponuja poceni rešitev za prenos majhnih količin podatkov, hkrati pa je tehnologija zelo energetsko varčna. Prav tako smo našli nekaj različnih raziskav ali člankov o meritvah porabe energije z uporabo padca napetosti preko referenčnega upora napajalnega kroga [3, 4, 5].

II Kratka zgodovina standarda Bluetooth

Standard Bluetooth je zasnoval dr. Jaap Haartsen leta 1994. Poimenovan je bil po danskem kralju, katerega so klicali Bluetooth. Leta 1998 je bila ustanovljena posebna skupina interesentov za standard Bluetooth (*Bluetooth Special Interest Group (SIG)*) in leta 1999 je bila izdana prva verzija standarda Bluetooth, imenovana Bluetooth 1.0. Čeprav uporablja isto frekvenčno območje kot Wi-Fi, je bil Bluetooth že od vsega začetka zasnovan kot tehnologija z veliko nižjo porabo energije in manjšim dosegom.

Leta 2002 je bila izdana različica 2.0, ki je prinesla nekaj zelo pomembnih izboljšav. Najpomembnejše izmed izboljšav so bile povečana prepustnost zaradi dodatka EDR (*Enchanced Data Rate*), povečan doseg na okoli 30 m ter za polovico manjša poraba energije. Verzija 2.1 je bila izdana leta 2007 in je prinesla poenostavljen proces vzpostavljanja povezave, kateri je bil hkrati tudi varnejši. Izboljšali so porabo energije za naprave, katere so večji del časa neaktivne. Leta 2009 je izšla verzija 3.0, ki je izjemno povečala prepustnost, saj je v kombinaciji z Wi-Fi povezavo omogočala hitrosti prenosa podatkov do 24 Mbps.

Bluetooth SIG je leta 2010 uradno izdal specifikacije verzije 4.0, ki prvič doda Bluetooth Low Energy tehnologijo. Le-ta je prinesla nove načine povezovanja, prav tako pa je prvič omogočila nekaj let delovanja naprav, ki uporabljajo gumbno baterijo. Leta 2013 je bila izdana verzijo 4.1, ki je prinesla veliko izboljšav v programskem delu, saj je omogočila razvijalcem več

kontrole nad parametri naprave, prav tako pa je omogočila direktno povezovanje naprav Bluetooth med seboj, brez posrednika.

Verzija standarda 4.2 je bila izdana leta 2014 in je prinesla več novosti, katere se s pridom uporabljajo še danes. Največja sprememba je bila povečanje maksimalne dolžine paketa, ki je prinesla približno desetkratno pospešitev hitrosti prenosa podatkov. Dodali so tudi možnost direktne povezave na internet ter izboljšali varnost povezave.

Standard Bluetooth 5 je bil uradno izdan leta 2016, vendar smo prve naprave dobili v letih 2017 in 2018. Verzija 5 je prinesla dvakratno hitrost prenosa podatkov, štirikrat daljši doseg (do 120 metrov) in osem krat večjo zmogljivost oddajanja. Hkrati se je izboljšalo sobivanje z drugimi brezžičnimi tehnologijami, ter poenostavilo poimenovanje, saj je sedaj uradno ime Bluetooth 5, brez decimalk.

III Bluetooth Low Energy

Standard *Bluetooth Low Energy (BLE)*, poznan tudi kot *Bluetooth Smart* je bil prvič predstavljen v verziji 4.0 standarda Bluetooth kot lahek del standarda Bluetooth, ki porabi izjemno malo energije. Čeprav si s klasičnim Bluetooth-om deli nekaj funkcionalnosti in ime, imata ta dva dela standarda povsem drugačne lastnosti in funkcionalnosti. Glavne razlike med različnimi verzijami specifikacij tehnologije BLE so predstavljene v Tabeli 3.1. *Bluetooth Low Energy* je razdeljen na tri glavne nivoje: krmilnik, gostitelj in aplikacija. Ti so naprej razdeljeni na dele, kot je razvidno s Slike 3.1. Večinoma so vsi trije nivoji implementirani na enem integriranem vezju, za nas pa so najpomembnejši deli fizični nivo (*Physical Layer - PHY*), generični profil dostopa (*Generic Access Profile - GAP*) in profil splošnih atributov (*Generic Attribute Profile - GATT*).

Fizični nivo je najnižji nivo sklada BLE in vsebuje analogno vezje za kodiranje, pošiljanje in prejemanje digitalnih simbolov preko elektromagnetnih valov. BLE oddajnik deluje v 2.4 GHz ISM frekvenčnem pasu. Standardi

BLE verzij v4.0, v4.1 in v4.2 pošiljajo podatke s hitrostjo 1 Mbps, standard BLE 5 pa doda še tri načine delovanja: visoko-hitrostni 2 Mbps ter dva načina za večji doseg s hitrostmi 125 kbps in 500 kbps. Zadnja dva načina prav tako uporabljata modulacijo s hitrostjo 1 Mbps, vendar je vsak bit zakodiran z večimi simboli, kar zniža prepustnost. To omogoča odpravljanje napak pri prenosu, kar pomeni, da se biti pravilno razpoznajo na večji razdalji. To hkrati pomeni, da za standard BLE 5 oglaševana dvakratna prepustnost in štirikraten doseg v resnici predstavljata dvakratno prepustnost ALI štirikraten doseg.

Povezovalni nivo (*Link Layer - LL*) je nivo, ki povezuje fizični nivo in gostiteljski del. Ker je zadolžen za upravljanje vseh časovnih omejitev je navadno ločen od višjih nivojev sklada. Ta nivo je zadolžen za oglaševanje, iskanje ter vzpostavitev in vzdrževanje povezav. Povezava v standardu BLE pomeni zaporedje prenosa paketkov ob vnaprej določenih časovnih intervalih. Čas med paketki je eden od parametrov povezave, ki se imenuje povezovalni interval (*connection interval*), skupek paketkov poslan ob tem času pa povezovalni dogodek (*connection event*). Ko je povezava vzpostavljena nivo LL skrbi za zanesljiv prenos podatkov na višjih nivojih. Povezovalni interval je ključen pri zagotavljanju nizke porabe energije, saj je oddajnik vključen le ob povezovalnih dogodkih, sicer pa je izključen, kar prihrani izjemno veliko energije.

Generični profil dostopa (*Generic Access Profile - GAP*) uporablja nižje nivoje za definiranje višjenivojskih vlog, procedur in načinov delovanja, ki omogočajo napravam prenos podatkov, odkrivanje servisov, upravljanje povezav in varnosti. To je temeljni profil naprav BLE, ki je obvezen za vse naprave. Nivo GAP v standardu BLE določa štiri vloge naprav, naprave pa lahko uporabljajo eno ali več vlog hkrati. Vlogi opazovanca in opazovalca se uporabljata za pošiljanje podatkov večim opazovalcem hkrati, ki niso povezani z opazovancem. Vlogi centralne in periferne naprave se uporabljata za prenos podatkov preko vzpostavljene povezave med dvema napravama. Centralna naprava podpira več hkratnih povezav na več perifernih napravah

in je tista naprava, katera vzpostavi in kontrolira povezavo.

Profil splošnih atributov (*Generic Attribute Profile - GATT*) definira načine, kako naprave BLE prenašajo podatke z uporabo konceptov imenovanih servisi in karakteristike. GATT definira kako se uporablja nižje nivoje za odkrivanje, branje, pisanje, oddajanje in sprejemanje podatkov. Podatki so organizirani hierarhično, v skupine imenovane servisi, kateri združujejo konceptualno povezane skupine podatkov imenovane karakteristike. Karakteristike si lahko predstavljamo kot zabojnike podatkov, vsaka karakteristika pa ima več atributov, kot so lastnosti karakteristike, vrednosti podatkov ter dodatni opisi karakteristike. Hierarhijo podatkov lahko vidimo na Sliki 3.3. GATT prav tako določa referenčno ogrodje profilov definiranih s strani Bluetooth SIG. Ti osnovni profili zagotavljajo skupno delovanje naprav različnih proizvajalcev. GATT definira dve vlogi naprav, GATT server, ki hrani podatke in GATT klient, ki se poveže na GATT server ter bere, piše in sprejema podatke. Najbolj pogosta konfiguracija naprav je GAP periferna naprava v vlogi GATT serverja in GAP centralna naprava v vlogi GATT klienta, vendar so možne tudi druge konfiguracije. Vsi GATT atributi, karakteristike in servisi se naslavljajo z uporabo edinstvenih identifikatorjev (UUID). GATT definira postopke za upravljanje s podatki in konfiguracijami naprav. Postopka branja in pisanja podatkov nadzoruje GATT klient. Za ta dva postopka je potrebna potrditev GATT serverja, kar se zgodi v naslednjem povezovalnem dogodku, ki sledi poslanemu zahtevku. Postopek pisanja podatkov brez potrdila začne GATT klient, vendar mu ni potrebno čakati na potrdilo GATT serverja, da lahko pošlje naslednji paketek. Postopka obveščanja in indikacije podatkov začne GATT server ob vnaprejšnji konfiguraciji s strani GATT klienta. Indikacija podatkov zahteva potrditev s strani GATT klienta, medtem ko obveščanje tega ne potrebuje, zaradi česar je prenos podatkov na ta način veliko hitrejši.

Strukturo komunikacijskega paketa standarda BLE lahko vidimo na Slikah 3.4 in 3.5. Velikost podatkov je odvisna od različice standarda BLE. V verzijah v4.0 in v4.1 je bila ta maksimalno 27 bajtov, od različice v4.2 na-

prej pa je 251 bajtov. Od tega je potrebno odšteti 4 bajte za glavo L2CAP nivoja, s čimer pridemo do 23 bajtov, oziroma 247 bajtov, kar je definirano v parametru povezave imenovanem *ATT_MTU*. Paketek nivoja ATT vsebuje še nadaljnje 3 bajte glave, s čimer pridemo do največjega števila podatkov za prenos na nivoju GATT, ki je 20 oziroma 244 za različico v4.2 in kasnejše različice.

IV Metodologija testiranja

Testirali smo s pomočjo dveh Nordic Semiconductor nRF52840 DK [6] razvojnih plošč, od katerih je bila ena konfigurirana kot periferna naprava na kateri se je merila poraba energije, druga pa kot centralna naprava, katera je upravljala s testi.

Za meritve smo uporabili *Power Profiler Kit* [7] proizvajalca Nordic Semiconductor. To je sistem, ki se priklopi na razvojno ploščo *nRF52840-DK* [6], katero uporablja za napajanje in komunikacijo z računalnikom. Ta sistem je razvit posebej za meritve porabe energije vgrajenih sistemov kakršne so naprave Bluetooth. Sistem ima analogno meritveno vezje, katero za zagotavljanje večje natančnosti razdeli območje merjenja od 1 µA do 70 mA na tri razpone, kar lahko vidimo na Tabeli 4.1. Meritveni sistem prikazuje meritve na računalniku, od koder meritveni sistem tudi krmilimo. Meritve so časovno omejene na maksimalno dolžino 120 s. Primer meritve z izbranimi podatki enega testa lahko vidimo na Sliki 4.1.

Porabo smo merili na GATT periferni napravi. Vezje smo nastavili tako, da smo vse dele vezja, ki niso nujno potrebni za delovanje povezave Bluetooth izklopili, saj bi drugače vplivali na rezultate meritev. Za lažjo sinhronizacijo testov in rezultatov smo na periferno napravo dodali 470 Ω upor, skozi katerega je v času med dvema testoma tekel električni tok, v samem trajanju testa pa je bil le-ta odklopljen. Na ta način je bilo v rezultatih meritev zelo enostavno opaziti kdaj se je test začel in končal, saj smo dosegli vidne preskoke porabe energije med testnim načinom delovanja in stanjem pripravljenosti.

Odločili smo se, da načinov delovanja standarda BLE 5 za zagotavljanje večjega dosega ne bomo testirali, saj ti načini niso namenjeni visoki pretočnosti podatkov. Pri razdaljah, ki jih ti načini omogočijo, navadni načini ne delujejo, tako da bi bila primerjava brezpredmetna, saj gre za povsem drugo namembnost. Zaradi časovne omejitve 120 s merilnega sistema smo pred testi izračunali teoretičen čas trajanja prenosa podatkov različnih testov, kar lahko vidimo v Tabelah 4.2 in 4.3. Zaradi (pre)dolgega trajanja prenosa podatkov pri nekaterih testnih parametrih, smo zasnovali dva sklopa testov. Pri prvem testu smo testirali neodvisnost povprečne porabe energije od količine prenesenih podatkov nad neko mejo, pri drugem pa porabo energije v odvisnosti od pretočnosti. Parametre testov lahko vidimo v Tabeli 4.2 za prvi in Tabeli 4.4 za drugi sklop testov.

Vsi testi so bili izvedeni petkrat in nato povprečeni z Enačbo 4.7. Vse meritve so bile izvedene preko celotne transakcije. Konec transakcije je bil definiran kot trenutek, ko lahko pričnemo naslednjo operacijo istega tipa na isti karakteristiki. Pri operacijah GATT branja in GATT pisanja to pomeni da so bile meritve izvedene preko dveh povezovalnih intervalov pomnoženo s številom paketkov. Za to smo se odločili, ker vsak zahtevek za GATT pisanje in branje potrebuje svoj GATT odgovor, katerega se pošlje v naslednjem povezovalnem intervalu, kar pomeni, da lahko naslednji zahtevek pošljemo šele v naslednjem povezovalnem intervalu po GATT odgovoru. Operaciji obveščanja in pisanja brez odgovora lahko izvajamo brez čakanja na GATT odgovor, zato smo pri teh operacijah kot konec transakcije definirali trenutek ko so vsi podatki uspešno prenešeni, četudi to ni na koncu povezovalnega intervala, saj lahko isto z isto operacijo nadaljujemo takoj za tem (v istem povezovalnem dogodku). Merjeni intervali in merjene količine za vse merjene Bluetooth operacije so povzete v Tabeli 4.5.

V Implementacija vgradne programske opreme

Vgradno programsko opremo (*firmware*) smo pripravili posebej za potrebe teh testov. Projekti za centralno in periferno napravo so bili ločeni, vendar so uporabljali iste module za povezovanje. Arhitektura *firmware-a* je vidna na Sliki 5.1. Zaradi izvajanja testov smo na centralni in periferni napravi morali uporabiti časovnik.

Centralno in periferno jedro sta modula na centralni in periferni napravi, ki vsebujeta vso potrebno logiko za izvajanje testov. To smo implementirali z uporabo avtomata stanj. V periferni napravi se avtomat stanj odziva na prejete ukaze s centralne naprave, v centralni napravi pa avtomat stanj poskrbi za povezovanje na periferno napravo, nakar čaka na pritisk gumba, kateri zažene teste. Testi so definirani v modulu testnih parametrov, kateri poskrbi tudi za gradnjo in preverjanje podatkov, kateri se pošiljajo med testi. Centralna in periferna BLE modula ter modul BLE abstrakcije skrbijo za visoko abstrakcijo funkcij potrebnih za izvajanje testov. Delujejo kot lepilo med komponentami sklada BLE, implementiranimi s strani Nordic Semiconductor, ter višjih modulov. Ti moduli poenostavijo operacije povezovanja, nastavlanja povezovalnih parametrov, branja, pisanja, obveščanja, in podobno, prav tako pa združijo dogodke prožene na nižjih nivojih v enoten format, katerega predstavijo modulu jedra. Komponente Nordic BLE Sklada so deli BLE sklada, kateri skrbijo za abstrakcijo nivojev kot so GATT, GAP, L2CAP in ostali. Nordic S140 SoftDevice je vnaprej pripravljena (pri Nordic Semiconductor) "črna škatla", katera se sklada s specifikacijami brezžičnega protokola BLE.

Periferna naprava vsebuje lasten servis z dvema karakteristikama, kot je vidno na Sliki 5.2. Karakteristika kontrole testov se uporablja za pošiljanje ukazov periferni napravi, kot so nastavljanje parametrov testa ter začetek in konec testa. Karakteristika testnih podatkov se uporablja za dejanski prenos podatkov med testom.

VI Rezultati in analiza

Slike 6.1, 6.2 in 6.3 prikazujejo primere meritev porabe za tri različne primere testov. Na sliki 6.1 lahko identificiramo posamezne pakete operacije pisanja, poslane v posameznih povezovalnih dogodkih. Programska oprema privzeto prikazuje skalo toka od $-100\,\mu A$ naprej, zaradi česar lahko izgleda da smo izmerili negativen tok, čeprav je bil dejanski izmerjen tok vedno nad $0\,\mu A$. Kot predvideno v teoretičnih izračunih, predstavljenih v Tabeli 4.2 je bilo za prenos 100 bajtov podatkov potrebnih 5 podatkovnih paketov, ki so zasedli 10 povezovalnih dogodkov in potrebovali 75 ms. Slika 6.2 prikazuje meritev porabe pri branju podatkov z različico v4.2 standarda BLE. Tudi operacija branja zahteva potrditev s strani GATT serverja, zato vsako branje podatkovnega paketa porabi dva povezovalna dogodka. Slika 6.3 prikazuje meritev porabe pri prenosu podatkov z različico BLE 5 z uporabo operacij obveščanja, za kar potrditve na nivoju GATT niso potrebne, temveč se opirajo le na potrjevanje nižjega povezovalnega nivoja. Zaradi tega je lahko poslanih več paketov v vsakem povezovalnem dogodku, kar izjemno poveča pretočnost podatkov. Vidimo lahko tudi, kdaj se začne vsak povezovalni dogodek, saj se pred tem oddajnik za kratek čas izklopi, da prepusti morebitnim ostalim povezavam prostor za komunikacijo. Meritvi porabe periferne naprave v primeru, kadar Bluetooth ni aktiven, sta prikazana na slikah 6.4 in 6.5. Slika 6.4 prikazuje meritev porabe z uporabljenim časovnik, slika 6.5 pa meritev porabe brez uporabe časovnika. Razlika porabe v obeh primerih je poraba časovnika in znaša približno $580\,\mu A$.

Prikaz odvisnosti porabe energije od količine podatkov lahko za operacijo pisanja vidimo na sliki 6.6 in za operacijo branja na sliki 6.7. Tu vidimo potrditev pričakovanj, da je porabljena energija sorazmerna s količino prenešenih podatkov in časom prenosa. Prenos vsakega podatkovnega paketa potrebuje dva povezovalna dogodka, kar pomeni da večje število podatkov enostavno potrebuje večje število podatkovnih paketov, ki v povprečju porabijo enako količino energije. Skupna energija za prenos neke količine podatkov povečuje v linearni odvisnosti od količine prenešenih podatkov. Zaradi

uporabe časovnika, kateri doda konstantno porabo prb. 580 µA pri večjih povezovalnih intervalih le-ta prevlada nad porabo Bluetooth oddajnika/sprejemnika. Na Slikah 6.8 in 6.9 lahko vidimo kakšna bi bila poraba brez uporabe časovnika. Razlike med porabo energije pri različnih časovnih intervalih so manjše, vendar zaradi neničelne porabe energije ko je BLE oddajnik/sprejemni ugasnjen poraba energije ni neodvisna od časovnega intervala, kot bi v teoriji morala biti (če bi bila poraba ničelna kadar naprava ugasne BLE oddajnik/sprejemnik).

Primarni načini spreminjanja pretočnosti v standardu Bluetooth Low Energy so spreminjanje parametrov ATT_MTU, povezovalnega intervala in hitrosti modulacije PHY nivoja. Ker je smiselno nastaviti največje parametre ATT_MTU ter PHY modulacije, ki jih naprave podpirajo, nam za spreminjanje pretočnosti ostane le še povezovalni interval. Slika 6.10 prikazuje spreminjanje pretočnosti ob različnih povezovalnih intervalih. Pretočnost prenosa se spreminja v skladu s pričakovanji - manjši kot je povezovalni interval, več paketkov lahko pošljemo v eni sekundi, zato bo pretočnost višja. Vidimo lahko tudi, da višji ATT_MTU občutno poveča pretočnost v različicah v4.2 in 5. Operacija obveščanja po drugi strani ni omejena z GATT odgovori, tako da bi pričakovali, da bo imela dolžina povezovalnega intervala manjši vpliv in da bo z daljšimi povezovalnimi dogodki tudi večja pretočnost, saj bo oddajnik lahko prižgan v celotnem povezovalnem intervalu. Slika 6.11 prikazuje rezultate pri operaciji obveščanja, kjer vidimo da nad neko dolžino intervala pretočnost prične padati. Do tega pride zaradi dejstva, da se v primeru neuspešnega prenosa paketa, povezovalni dogodek zapre. Takrat se lahko pošiljanje nadaljuje šele ob naslednjem intervalu. To pomeni, da bo ob večjem intervalu tudi povprečen čas, ko je dogodek zaprt, daljši, zaradi česar pretočnost pade. Ob primerjavi s Tabelo 6.1 lahko vidimo, da so dosežene hitrosti pretočnosti zelo blizu teoretični meji za vsako različico standarda BLE.

Ker sta operaciji branja in pisanja omejeni z odgovori GATT serverja so rezultati obeh operacij praktično isti. Ob pogledu na Sliko 6.12 vidimo, da

je uspelo razvijalcem standarda Bluetooth z vsako novo različico zmanjšati porabo moči za prenos podatkov glede na pretočnost. To jim je uspelo kljub temu, da različica BLE 5 deluje pri višji hitrosti. To omogoča dejstvo, da se pri višjih hitrosti podatki hitreje prenesejo, kar pomeni, da se lahko oddajnik prej izklopi. To je lepo razvidno na Sliki 6.13, ki kaže skupno energijo potrebno za prenos določene količine podatkov v odvisnosti od pretočnosti. Vidimo da je najbolj energijsko učinkovit prenos pri najvišjih pretočnostih, pri čemer je potrebno poudariti, da je pri merjenju različice v4.1 količina prenesenih podatkov manjša, prikazana pa je ekstrapolacija za isto količino podatkov.

Zaradi istih lastnosti je analiza operacij obveščanja in pisanja brez potrdila združena. Ker ni potrebno čakati na odgovor druge naprave, je možno opraviti več operacij v enem povezovalnem dogodku. Na Sliki 6.14 je razvidno da različici v4.1 in v4.2 uporabljata isti fizični nivo, saj sta največja in najmanjša moč praktično enaki, vendar pri različnih pretočnostih. Vidimo, da hitrejša modulacija standarda BLE 5 potrebuje več energije, vendar je zato tudi dosti večja pretočnost podatkov. Bolj zanimiva je Slika 6.15, kjer vidimo velik preskok skupne porabe energije za prenos iste količine podatkov z vsako različico standarda. Za oba preskoka zmanjšanja porabe energije je odgovorno dejstvo, da se lahko oddajnik hitreje izklopi, pri čemer je razlog pri različici v4.2 povečanje količine podatkov v posameznem paketku, pri različici 5 pa višja hitrost modulacije. Zanimivo je, da je količina energije potrebne za prenos neke količine podatkov praktično neodvisna od pretočnosti. Ta informacija je zelo pomembna, saj nam omogoči, da parametre povezave nastavimo glede na potrebe, brez posebne skrbi za porabo energije pri večjih količinah prenešenih podatkov.

VII Sklep

Nedavno predstavljena različica 5 standarda Bluetooth obljublja dvakratno hitrost, štirikraten doseg in osemkrat boljšo zmogljivost oddajanja, ki naj ne

bi vplivala na porabo moči. V magistrski nalogi je predstavljeno opravljeno delo potrebno za meritve in primerjavo energetske učinkovitosti tehnologije Bluetooth Low Energy (BLE). Predstavljene so primerjave porabe električne energije glede na pretočnost med različnimi verzijami specifikacije BLE.

Da smo premostili omejitve merilnega sistema, smo morali najprej potrditi, da je povprečna poraba neodvisna od količine prenešenih podatkov. Nato smo testirali vpliv povezovalnega intervala na pretočnost, kjer smo ugotovili da je pretočnost najvišja ob najnižjem intervalu za operacije branja in pisanja. Za operacije obveščanja in pisanja brez potrdila je za najvišjo pretočnost potrebno najti ravnovesje med velikostjo intervala ter zasedenostjo radijskih valov v danem okolju. Z meritvami potrebne moči in porabe energije za prenos podatkov smo ugotovili, da se energetska učinkovitost standarda Bluetooth izboljšuje z vsako različico. Ugotovili smo, da je za najnižjo porabo energije v vseh primerih potrebno najti najvišjo pretočnost, saj se v tem primeru lahko oddajnik najhitreje izklopi, s tem pa se tudi zmanjša poraba energije. Hitrejši prenos podatkov hkrati pomeni tudi manjšo obremenitev brezžičnega okolja, kar zmanjša število trčenj paketov in s tem tudi količino večkrat poslanih paketov, kar še dodatno zmanjša porabo energije.

Rezultati in ugotovitve predstavljene v tej magistrski nalogi dokazujejo, da ni razloga da nadaljnji razvoj naprav Bluetooth ne bi uporabljal različice Bluetooth 5, saj le-ta prinese višjo pretočnost, daljši doseg in boljše sobivanje z ostalimi brezžičnimi tehnologijami, saj vse to zagotavlja pri nižji porabi energije za prenos podatkov.

Chapter 1

Introduction

In recent years, we have seen significant progress in digital technologies, that contribute to the ever-present concept of "Internet of Things" (IoT). Since the vast majority of IoT products use Bluetooth for communication, it is still evolving and is continuously improved. In our company, we are working on the development of a new product that uses Bluetooth Low Energy for communication. During development we have encountered a problem, where we needed a study, that would compare energy consumption in relation to throughput with different connection parameters applied and across different Bluetooth Low Energy versions. This is especially interesting for the new Bluetooth 5, which promises twice the throughput, four times the range and eight times increased broadcast capacity, all of which is supposed not to affect power consumption [8].

1.1 Related Work

Comparing energy consumption when transferring data using different technologies is nothing new. To the best of our knowledge, there is no such study, that compares different versions of Bluetooth technology. Such comparison would be even more interesting with the arrival of Bluetooth 5 [8], which introduces many innovations that are incompatible with previous versions

without modifying the physical level of the device.

A comparison of the power consumption between Bluetooth 4.0 Low Energy and ZigBee / 802.15.4 can be found in the article "How Low Energy is Bluetooth Low Energy? Comparative Measurements with ZigBee/802.15.4" [1], that supersedes older articles that compare Bluetooth Classic (2.0) with other technologies. The authors also test the effect of interference on Bluetooth and ZigBee technology. For Bluetooth Low Energy measurements, they were using a BlueGiga BLE112 module (based on TI CC2540 SoC), which was configured as a slave peripheral device, as the device under test for which power consumption was measured. The module was running Texas Instruments's BLE Stack v1.0 in single chip mode. This represented a realistic example of a sensor where the amount of hardware and power consumption is to be kept at a minimum. Since they were using BLE version 4.0, the maximum application layer packet payload size is 20 bytes, which has a measurable effect on power consumption. They were using the Monsoon Power Monitor [9] for power measurements, which uses the technique of measuring the voltage drop through the shunt resistor. The authors concluded that BLE (version 4.0) indeed consumes a comparatively small amount of energy and has a very attractive ratio of energy per bit transmitted.

Another study comparing BLE with other wireless technologies in regards to power consumption was made for the article "Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI" [2]. The authors used Texas Instrument CC2540 System-on-Chip as their BLE device under test and used TI BLE Stack v1.2.1, which implements Bluetooth Low Energy specification version 4.0. They performed their tests in the laboratory environment with the peripheral and central devices placed one meter apart. They used the current-shunt resistor method for measuring the power consumption of the device under test. The results presented in that paper revealed that BLE v4.0 already provided an inexpensive and power-efficient solution for wireless communication, even if it had several downsides, which were resolved in later BLE versions, when compared to other wireless

technologies.

Although the article "An Experimental Performance Evaluation and Compatibility Study of the Bluetooth Low Energy Based Platform for ECG Monitoring in WBANs" [10] was released when Bluetooth Low Energy v4.2 was already released, the authors still used v4.0/v4.1 for their experiments. Their test datasets were specific to healthcare applications, such as transmitting ECG data wirelessly. They performed their measurements using the BLE112 wireless module, running a BGScript application, supported by BlueGiga.

The technique of measuring the voltage drop through the shunt resistor was used for power consumption measurements in related articles. Power consumption measurements using the shunt resistor technique are described in multiple articles, such as "Measuring Bluetooth® Low Energy Power Consumption" [3], "Power Measurements Techniques For Embedded Systems" [4] and "Embedded Systems Power Consumption Measurement Methods Overview" [5]. The current-shunt resistor technique is commonly used for power consumption measurements. It uses a high precision (better than 1%), low impedance (usually less than $1\,\Omega$, but can be more) resistor, placed in series with current flow to, or from the device under test. The scheme of this technique is presented in Figure 1.1. The voltage drop over the resistor is measured, from which we can calculate the current using the Ohm's law equation. It is important to select a high precision resistor, as this will make our measurements and calculations more precise. Selection of resistor value is usually below $1\,\Omega$, as this will have a small voltage drop, which has a small impact on the measured circuit. A higher value may be selected as well if the power supply voltage and minimum voltage needed by the measured circuit allow it. Selecting a larger value will result in a larger voltage drop, which is easier to measure and is thus more appropriate for measurements of small currents. All three mentioned articles describe other current measurement methods, such as the Charge transfer method, which uses a capacitor, where charging or discharging time is measured to measure the current, the Current mirror method, which uses four bipolar transistors and the Current

probe method, which exploits magnetic coupling of currents running through wires.

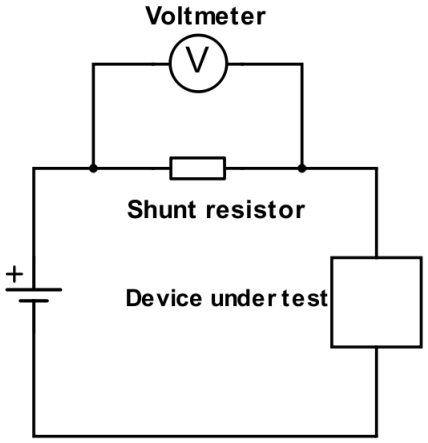

Figure 1.1: Current shunt resistor measurement scheme.

1.2 Thesis content

In Chapter 2, we describe the history of Bluetooth and Bluetooth Low Energy, as well as general differences between the Bluetooth standard versions and when those improvements were introduced.

In Chapter 3, we describe the Bluetooth Low Energy stack, what are the responsibilities of each layer and how layers interact with each other. The properties and mechanisms of the layers that have major impact on throughput and power consumption are given in detail.

In Chapter 4, we describe the hardware and software tools used for power consumption measurements. Measuring system and hardware configuration are explained and the test scenarios that were used as the basis for power consumption measurements are presented.

In Chapter 5, we describe the tested device firmware implementation development. Developed layers are presented and the testing firmware mech-

anisms are explained for both the peripheral and central devices.

In Chapter 6, we describe measurement examples and results for different tests. We analyzed the relationship between energy consumption and amount of data transmitted, the data throughput and power consumption of BLE standards for different connection parameter settings and diferent BLE standards.

Finally, we conclude the thesis in Chapter 7, recapping the work done and present the most important findings.

Chapter 2

A brief history of Bluetooth

2.1 Version 1.0

The Bluetooth standard was initially conceived by Dr Jaap Haartsen at Ericsson back in 1994. It was named after the Danish king Harald Blatand, nicknamed Bluetooth, who united Denmark, Norway and Sweden in the 10th century. At the time Ericsson wanted to replace the wired standard RS-232 with a wireless alternative, while other companies including Intel and Nokia also had the idea of wirelessly linking cell phones and computers around the same time.

In 1998 the Bluetooth Special Interest Group (SIG) was formed with five companies: Ericsson, Nokia, Intel, Toshiba and IBM. The Bluetooth SIG still to this day publishes and promotes the Bluetooth standard and its subsequent revisions, and although it started as a group of five companies, it reached 4,000 members by the end of its first year. The group now contains over 33,000 member companies at various levels of influence.

Although Bluetooth uses a very similar frequency bandwidth (2.4 - 2.485 GHz) as Wi-Fi, it has been designed as a much shorter range and lower power alternative from the start. Version 1.0, released in 1999, proposed as a replacement for the RS-232 standard, was designed as a flexible packet-based protocol with a wide selection of profiles. The standard came with profiles

for wireless voice and headsets, dial-up networking, fax and file transfers, which were the primary use cases for RS-232, but the set of profiles has increased substantially since then. The first version had some problems, such as anonymity issue due to compulsory address broadcasting, connection problems, slow (theoretical) data speeds of just 721 kbps and short range of about 10 meters.

2.2 Version 2.0 + EDR

In 2002 IEEE approved 802.15.1 specification to conform with Bluetooth wireless technology and two years later Version 2.0 [11] with Enhanced Data Rate (EDR) was released. The same year Bluetooth technology reaches an installed base of 250 million devices and product-shipment rate surpasses 3 million per week. Version 2.0 brings some much-needed improvements, such as increased range of around 30 meters, an increased theoretical core connection speed of 1 Mbps and 3 Mbps with EDR. Power consumption was also reduced by 50% in comparison to the previous version.

2.3 Version 2.1 + EDR

In 2007 Bluetooth 2.1 [12] standard was released. Secure Simple Pairing (SSP) was the most important feature of this version, which made the pairing process more secure and simpler. SSP made encryption mandatory for all connection, which made man-in-the-middle attacks more difficult. Another important improvement brought in version 2.1 was sniff subrating, which improved battery life in devices that are inactive most of the time, such as keyboards and headsets, by reducing the active duty cycle of Bluetooth devices.

2.4 WiBree acquisition

In June 2007 the Bluetooth SIG acquired the Wibree Alliance, an initiative led by Nokia (together with Nordic Semiconductor, Broadcom Corporation, CSR and Epson) to develop an ultra-low power form of wireless connectivity, which uses much less power than existing Bluetooth technology. Nordic and other brought their knowledge of ultra-low power wireless connectivity to the Bluetooth SIG, enabling it to develop what will become Bluetooth Low Energy (BLE).

2.5 Version 3.0 + HS

Version 3.0 High Speed (HS) [13] specification was released in 2009, with the key feature being high-speed data transfer. In conjunction with 802.11 Wi-Fi radio, it could reach data speeds of up to 24 Mbps. It used the 802.11 link for transferring large amounts of data while still using Bluetooth radio for discovery, connection and configuration, which made it less power hungry than classic 802.11 Wi-Fi link. This version also permitted near-field communication (NFC) for pairing.

2.6 Version 4.0 + LE

The Bluetooth SIG announces the formal adoption of Bluetooth Core Specification Version 4.0 with Low Energy [14] technology in 2010. This version defines two chips, one for Classic Bluetooth v4.0 (BR/EDR) and one for Bluetooth Low Energy (also called Smart Bluetooth). The Bluetooth Low Energy technology was heavily based on WiBree. Adoption of BLE in Bluetooth Core Specification v4.0 by Bluetooth SIG made it possible for BLE to complement the existing Classic Bluetooth, while still making it possible to run it off a single coin cell battery. Bluetooth LE was mainly designed to frequently transmit small chunks of data to and from small devices while

saving power. That empowered a new way to gather data from various sensors, which enabled the development of heart rate monitors, thermometers, health and fitness trackers, etc., which all took advantage of the new low power feature.

Aside from Bluetooth LE feature, this version also introduced the Security Manager (SM) services with AES encryption and the Generic Attribute Profile (explained in Chapter 3.5), which is the backbone of a Bluetooth LE device, as it provides the profile of the device.

2.7 Version 4.1 LE

Version 4.1 [15], introduced in 2013 brought very few changes in hardware, speed and range, but many improvements on the software side. Bluetooth 4.1 focused on implementing the groundwork to drive IoT devices. It could indirectly connect devices to the cloud that would have otherwise been out of range. Another improvement was coexistence with 3G/LTE radios, as previous versions had problems such as poor performance and battery draining when used simultaneously. Even though Bluetooth and 3G/4G radios use different frequency bands, the frequencies used are adjacent to each other, which meant that radios in mobile devices, which are physically placed close to each other or even implemented on the same chip, brought increased interference when Bluetooth and 3G/4G radios were used at the same time. Bluetooth 4.1 eliminates this problem by coordinating its radio with 3G/4G radios automatically so there is no overlap in the time domain, which means that both Bluetooth and mobile network radios can perform at their maximum potential. This version also allowed developers to have more control over connections, making it easier to manage devices power consumption. Additionally, Version 4.1 made it possible to treat any device as both a peripheral and a hub at the same time, enabling the direct connection between peripheral devices without an intermediary.

2.8 Version 4.2

Version 4.2 [16] was released in 2014, and it brought multiple improvements and is still the most widely used standard today. The most notable improvement was the LE Data Length Extension (DLE), which increased the capacity of each packet from 20 bytes to 244 bytes. It also enabled devices to use Internet Protocol version 6 (IPv6) for direct internet access, which allowed sensor and smart devices transmitting data directly over the internet. Another improvement was LE Secure Connections, which made it harder to track devices without permission.

2.9 Version 5

Bluetooth version 5 [17] was officially adopted by Bluetooth SIG in 2016, with most of the major smartphone manufacturers adopting it in 2017 and 2018. Its purpose is to optimise wireless technology by increasing the functionality of Bluetooth. Aside from improvements involving IoT, Bluetooth 5 promises two times faster transfer speed, four times longer range (up to 120 meters) and eight times the data broadcasting capacity. The improvements made to support these claims are explained in Chapter 3. A new and improved Channel Selection Algorithm #2 (CSA #2) is added, which improves the interference tolerance and multi-path fading effects of the Bluetooth radio, as well as allows the radio to limit the minimum number of channels used for channel hopping in high interference environments. When compared with CSA #1, which was used only for channel selection for connection events, CSA #2 supports channel selection for periodic advertising packets as well. This improves coexistence between different Bluetooth connections as well as with other wireless technologies, which is becoming more and more critical as we enter a complex world of IoT smart devices. Bluetooth SIG dropped the point number of the name, so it is now just called Bluetooth 5 (instead of Bluetooth 5.0 or 5.0 LE), which is supposed to ”simplify marketing and communicate user benefits more effectively” [18].

Chapter 3

Bluetooth Low Energy

Bluetooth Low Energy (BLE), also known as Bluetooth Smart, was introduced in Bluetooth Core Specification Version 4.0 [14] as a light-weight, ultra-low power subset of classic Bluetooth. While there is some functionality overlap with classic Bluetooth and they share the name "Bluetooth", BLE has an entirely different property and functionality set. BLE started its life in 2001, in Nokia labs as an in-house project named WiBree, before it was adopted by Bluetooth SIG and renamed to Bluetooth Low Energy.

This chapter will give a quick overview of BLE, specifically how data is organised in BLE and how devices advertise their presence so that other devices can find and connect to them.

The biggest differences between versions of the Bluetooth Low Energy core specifications can be observed in Table 3.1.

3.1 BLE Stack

The Bluetooth Low Energy stack is split into three main layers, Controller, Host and Application, which in turn consist of multiple parts, as seen in Figure 3.1. In most cases those three layers are implemented on a single SoC, but they can also be distributed onto multiple physical SoCs, with most commonly split between the Host and the Controller [19]. For us, the

BLE Version	*ATT_MTU*	Data Rate	Other
4.0	23	1 Mbps	First release
4.1	23	1 Mbps	Improved 3G/LTE coexistence, Direct connections between devices
4.2	247	1 Mbps	IPv6, LE Secure Connections
5	247	125 kbps (Coded), 500 kbps (Coded), 1 Mbps, 2 Mbps	8x broadacting capacity, Range up to ~120m (Coded), Improved channel selection algorithm

Table 3.1: Biggest differences between BLE versions.

key elements of the BLE stack are the Physical Layer (PHY), the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT).

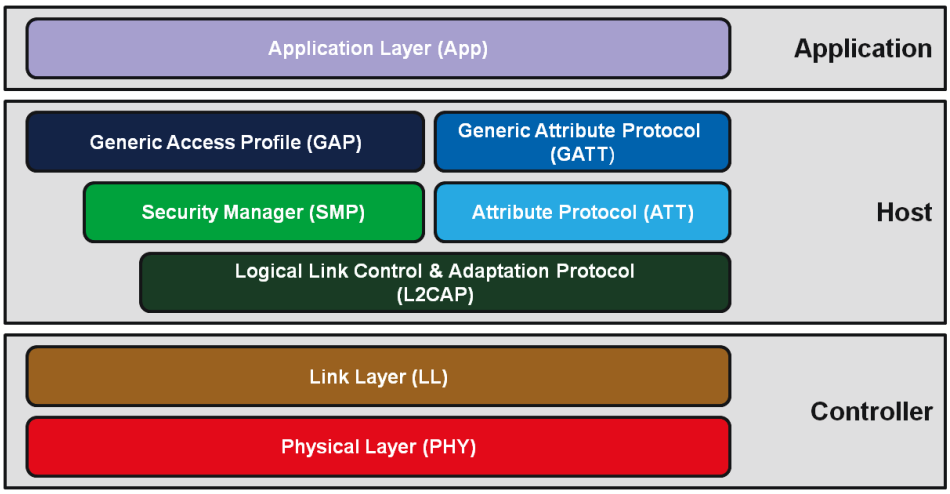

Figure 3.1: Bluetooth Low Energy stack[1].

[1]Image via Microchip: `http://microchipdeveloper.com/wireless:ble-introduction`

3.2 Physical Layer (PHY)

The BLE physical layer is the lowest layer of the BLE stack and contains the analogue circuitry responsible for translation, transmission and receiving of digital symbols over the air. BLE radio operates in the 2.4GHz ISM (Industrial Scientific Medical) band. A frequency hopping transceiver is used to combat interference and fading. The BLE system uses 40 RF channels between 2400 MHz and 2483.5 MHz with 2MHz spacing. The 40 channels are divided into 3 Advertising Channels and 37 Data Channels [20].

BLE v4.0, v4.1 and v4.2 used a radio, transmitting at 1 Mbps with one symbol per bit, using the 1 Msym/s GFSK (Gaussian minimum shift keying) modulation scheme with FHSS (Frequency-hopping spread spectrum) spreading, implemented as Adaptive frequency-hopping spread spectrum (AFH). With the BLE 5 Core Specification [17], Bluetooth SIG added two new modes: High Speed (HS) and Long Range (LR). The basic 1 Msym/s GFSK modulation scheme is still supported, as that is the only modulation scheme that is mandatory in a BLE device and is therefore used for backwards compatibility. The BLE 5 HS standard brings the possibility of using the 2 Msym/s GFSK modulation scheme, which doubles the transmission speed to 2 Mbps. BLE 5 LR (also known as coded) uses the same 1 Msym/s physical GFSK modulation scheme as older versions, but each bit is encoded at either 125 kbps (S=8) or 500 kbps (S=2).

It means that the "2x speed and 4x range" advertised by Bluetooth SIG for Bluetooth 5 actually means 2x speed OR 4x range, as those two options are on opposite sides of the spectrum, meaning we cannot have both at the same time.

3.3 Link Layer (LL)

The Link Layer is the part that ties together the PHY and the Host stack. It is usually implemented as a combination of custom hardware and software. Because it is responsible for managing all timing requirements, it is

Input (from FEC encoder)	Output with S=2	Output with S=8
0	0	0011
1	1	1100

Table 3.2: FEC Pattern Mapper behaviour

usually kept isolated from the higher layers of the stack, as it is the only hard real-time constrained layer of the whole protocol stack. It is responsible for advertising, scanning, and creating/maintaining connections, as well as coding the data before transmission, controlling the frequency hopping and acting as a reliable data barrier. Coding in Bluetooth 5 LR consists of two steps: Forward Error Correction (FEC) and pattern mapping. The convolutional FEC encoder uses a non-systematic, non-recursive rate $\frac{1}{2}$ code with constraint length $K = 4$. This means that it replaces a single bit with two encoded bits, before sending it to the pattern mapper. The pattern mapper converts each bit from the FEC into one (S=2) or four (S=8) symbols, as shown in Table 3.2, before sending.

At the link layer, the following role pairs are defined:

- Advertiser / Scanner (Initiator)
- Slave / Master
- Broadcaster / Observer

Once the Scanner has acquired enough information about the Advertisers in its range to decide to which to connect it, it becomes an Initiator, initiating a BLE Link Layer connection process. A connection is simply a sequence of packet transmissions between two BLE devices at predefined times. Once connected, the Link Layer acts as a reliable data barrier, meaning any data presented to the upper layers have been reliably transmitted. All packets are checked against a 24-bit CRC. If the check fails, re-transmissions are requested by the Link Layer until the packet is transmitted successfully, with the retries being done in the next connection event and channel.

3.3.1 Connection Events (CE) and Connection Interval (CI)

Once connected, the Master/Slave exchange data packets at regular intervals, called *connection events*, as seen in Figure 3.2. The *Connection Interval* can be set between 7.5 ms and 4000 ms with a step size of 1.25 ms. When in a connection, the master has to transmit a packet to the slave once every connection event. If there is no actual data to send, a 0-byte packet is sent to keep the connection alive. A connection event is the start of a group of data packets that are sent from the master to the slave and back again. The connection event continues until a packet either fails to be received correctly, the sender is satisfied with ending the connection event, or, in an improbable scenario, the end of the interval has been reached. If a packet fails to be received correctly, the connection event is ended, and the failed packet will be retried at the next connection event.

Connection Events and Connection Intervals are the concepts that enable the Bluetooth Low Energy to actually be Low Energy. That is because between CIs, the radios can be turned off, which saves tremendous amounts of energy. The most power-hungry part of a BLE chip is undoubtedly the 2.4GHz oscillator needed for the radio, and if we can keep the radio turned off as much as we can, the energy consumption will be lowered substantially. The CEs/CIs are thus used to turn on the radio only at predetermined intervals for short amounts of time, keeping the average power consumption low.

3.3.2 Connection parameters

Other than the *connection interval*, explained in Chapter 3.3.1, there are two more connection parameters managed by LL: *Slave latency*, which declares the number of connection events that a slave can choose to skip without risking disconnection, and *Connection supervision timeout*, which is the max-

[2]Image via Microchip: `https://microchip.wdfiles.com/local--files/wireless:ble-link-layer-connections/connected-phase.png`

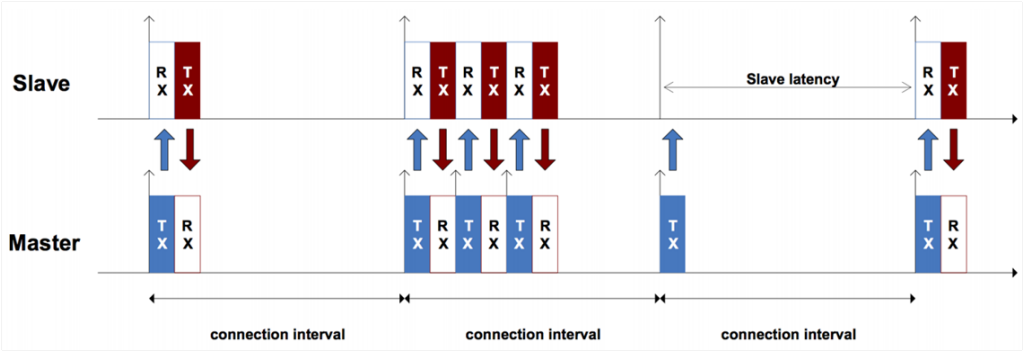

Figure 3.2: Connection events and connection intervals[2].

imum time between two received valid data packets before a connection is considered lost.

The connection parameters are negotiated at the initial connection procedure, but they can also be changed during the active connection. This can be used to dynamically increase throughput for transfer of larger chunks of data, while still maintaining low power consumption on average. This can be achieved by decreasing the connection interval for the data transfer and increasing it back up after the data was sent. That way, each connection can be fine-tuned to provide the best balance between throughput and power consumption.

3.4 Generic Access Profile (GAP)

Generic Access Profile (GAP) covers the usage mode of the lower-level radio protocols to define roles, procedures and modes that allow devices to broadcast data, discover devices, manage connections and negotiate security levels. This is a Bluetooth base profile which is mandatory for all Bluetooth devices.

3.4.1 Roles

In BLE, GAP defines four specific roles: Broadcaster, Observer, Peripheral, and Central. A device can operate in one or more BLE GAP roles at a time.

Broadcaster role is optimised for transmit-only applications that distribute data regularly. Broadcasters send data in advertising packets instead of in data packets, so that data is immediately accessible to any device listening, without the need for a specific connection to the Broadcaster.

The Observer is optimised for receive-only applications that want to collect data from broadcasting devices, without the need to form a connection with them. An Observer listens to data embedded in the advertising packets.

Central role corresponds to the Link Layer master and supports multiple connections. It is the initiator and controller of connections to peripheral devices. The device supporting the central role needs access to more processing and memory resources to be able to manage multiple connections and generally supports more complex functions compared to other BLE GAP devices. The central starts by scanning for other devices advertising packets and then initiates a connection with the selected device.

Peripheral role corresponds to the Link Layer slave. It is optimised for devices that support a single connection and are less complex than central devices. The peripheral device starts with advertisements to allow the central devices to find and establish a connection with it.

It is common for developers to associate BLE GATT *client* and *server* roles (explained in Chapter 3.5) with GAP *central* and *peripheral* roles, even though there is no connection between them, as any combination is allowed and used in different devices and use cases.

3.4.2 Modes and procedures

GAP layer is responsible for the following procedures, which are applicable to specific roles [17]: broadcasting, observing, advertising, discovery, connection establishment, connection parameter update, connection termination

and many more.

Advertising packets are blindly sent at fixed intervals unidirectionally, and they make up the basis for broadcasting (and observing), as well as discovery. An observing or scanning device may receive the advertising packet if it happens to scan while the advertising packet is transmitted, after which it can initiate the connection. A connection requires two devices that synchronously perform data exchanges at set regular intervals and provide guarantees on data transmission.

3.5 Generic Attribute Profile (GATT)

Generic Attribute Profile (GATT) defines the way BLE devices transfer data between themselves, using concepts called *services* and *characteristics*. GATT defines how to use Attribute Protocol (ATT) as its transport protocol to discover, read, write, broadcast and obtain indications of data. It is used by the application for data communication between two connected devices. The data is organised hierarchically in groups called *services*, which group conceptually related pieces of data called *characteristics*.

Attributes are the smallest data item defined by GATT (and ATT). They are addressable pieces of information that can contain user data or metadata. Metadata houses information about the structure and grouping of the different attributes contained within the server.

GATT also provides the reference framework for all GATT-based profiles, which are defined by Bluetooth SIG. These profiles cover precise use cases and ensure interoperability between devices from different manufacturers and range from mandatory ones, such as *Device Name*, to more use case specific such as *Heart Rate Measurement*.

3.5.1 Roles

As with any other profile or protocol in Bluetooth specification, GATT starts with defining the roles that devices can implement:

The GATT Server contains the data to be monitored or changed, organised in *attributes*. It receives requests from a client and sends back responses. It can also send server-initiated updates when configured to do so.

The GATT Client sends requests to a server and receives responses (and server-initiated updates) from it. Because the GATT client inherently has no information about the server's attributes, it must first inquire about those attributes by performing service and characteristic discovery. After completing service and characteristic discovery, it can then start reading and writing attributes (organised in services and characteristics) supported by the server.

While the most common configuration is a smaller device configured as GAP Peripheral and GATT Server, and a second device (most often a smartphone) configured as GAP Central and GATT Client, it is by no means mandatory, as already mentioned in Chapter 3.4.1.

3.5.2 UUIDs

A universally unique identifier (UUID) is a globally unique 128-bit (16-byte) number that is used to identify Profiles, Services and Data Types in a Generic Attribute (GATT) Profile. BLE specification adds support for shortened 16-bit and 32-bit UUIDs. These shortened formats can only be used with UUIDs that are defined by the BLE specification (i.e., that are listed as standard Bluetooth UUIDs by the Bluetooth SIG), while vendor-specific UUIDs always use 128-bit.

3.5.3 Attribute and Data Hierarchy

The data in a GATT server is grouped into *services*, each of which can contain zero or more *characteristics*, as seen in Figure 3.3. Each service distinguishes itself from other services by a UUID. Any device claiming GATT compatibility (essentially, all BLE devices sold) has to strictly follow this hierarchy

pattern.

Characteristics can be thought of as user data containers. They always include at least two attributes: the characteristic declaration (metadata) and the characteristic value (user data). They can also contain additional attributes called *descriptors*, which expand the metadata contained in the characteristic declaration. The most common descriptor used is the *Client Characteristic Configuration Descriptor (CCCD)*, which provides a mechanism for Server-Initiated updates (explained in Chapter 3.5.4).

Attributes are the smallest addressable data entity defined by GATT (and ATT). The most common attributes are:

- The attribute **handle** is a unique 16-bit identifier for each attribute on a particular GATT server.
- The attribute **type** is simply the UUID (see Chapter 3.5.2) of the characteristic.
- **Permissions** are metadata that specify which operations can be executed on each particular attribute. ATT and GATT define the following permissions:
  - Access Permissions: *None, Readable, Writeable, Readable and Writeable.*
  - Encryption: *No encryption, Unauthenticated encryption, Authenticated encryption.*
  - Authorization: *No authorization required, Authorization required.*
- **Value** holds the actual data content.

3.5.4 Features

GATT features are strictly defined procedures that allow data exchanges to take place using the GATT layer. There are 11 features defined in the GATT

[3]Image via Adafruit: `https://learn.adafruit.com/assets/13828`

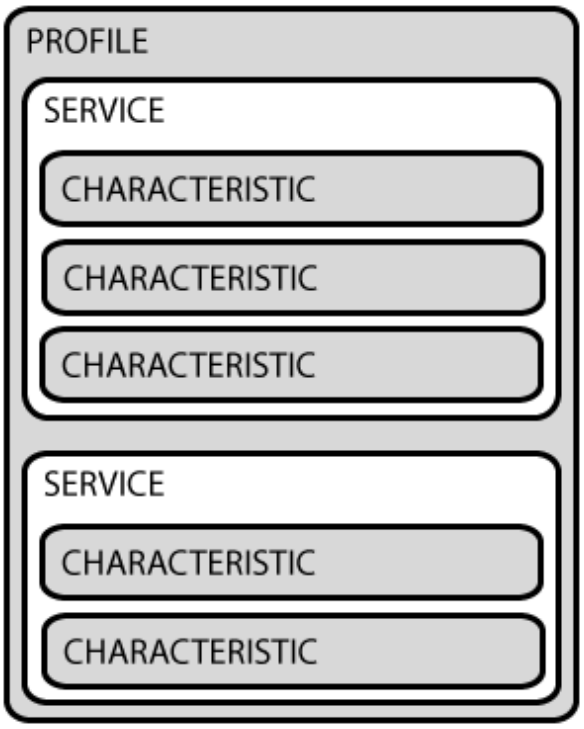

Figure 3.3: GATT data hierarchy[3].

Profile: *Server Configuration*, *Primary Service Discovery*, *Relationship Discovery*, *Characteristic Discovery*, *Characteristic Descriptor Discovery*, *Reading a Characteristic Value*, *Writing a Characteristic Value*, *Notification of a Characteristic Value*, *Indication of a Characteristic Value*, *Reading a Characteristic Descriptor* and *Writing a Characteristic Descriptor*. Here, the features relevant for this work, will be explained in more details.

Server Configuration: Exchange MTU

This (sub-)procedure is used by the client to set the *ATT_MTU* parameter (see Chapter 3.5.5) to the maximum value that is supported by both devices. The client will send an *Exchange MTU request*, to which the server will reply with either *Exchange MTU Response* or *Error Response*. The *ATT_MTU* parameter defines the largest data payload allowed by the ATT layer, which

means that the maximum data payload for GATT operations is $ATT\_MTU - 3$ bytes, as 3 bytes are used for the header.

Service and Characteristic Discovery

Because the client by itself has no knowledge about the server attributes, services and characteristics themselves and their handles it usually issues a series of sub-procedures to gather that information when it first connects to the server. These sub-procedures return the list of primary services and their handles, which can then be used to issue characteristic discovery for each returned service.

Characteristic Value Read

This procedure is used by the client to read a Characteristic Value from a server. Four sub-procedures can be used to read a Characteristic Value: *Read Characteristic Value*, *Read Using Characteristic UUID*, *Read Long Characteristic Values*, and *Read Multiple Characteristic Values*.

The *Read Response* to the *Read Characteristic Value* contains the complete Characteristic Value if that is less than $ATT\_MTU - 3$ bytes in length. Otherwise, it only contains the first part, and the *Read Long Characteristic Value* can be used to obtain the rest of the value.

The *Read Long Characteristic Values* sub-procedure request is issued with increasing offset as a request parameter until the reply is shorter than $ATT\_MTU - 3$ bytes.

Characteristic Value Write

This procedure is used by the client to write a Characteristic Value to a server. Five sub-procedures can be used to write a Characteristic Value: *Write Without Response*, *Signed Write Without Response*, *Write Characteristic Value*, *Write Long Characteristic Values* and *Reliable Writes*.

Write Characteristic Value can be used by the client to write a Characteristic Value if it is smaller than $ATT\_MTU - 3$ bytes and the server will

acknowledge the operation with a response.

When the value to be written is larger than $ATT\_MTU - 3$ bytes, the client can use *Write Long Characteristic Values*. Here the data (up to 512 bytes long) is split into multiple *Prepare Write Requests*, each of which can have at most $ATT\_MTU - 5$ bytes in its payload, as two bytes are needed to transfer the offset. Each *Prepare Write Request* will get a *Prepare Write Response* from the server, which sends the offset, as well as data for the client to confirm. After all the data is transferred, an *Execute Write Request* is used to write the complete value.

Reliable Writes sub-procedure is used when the client wants to write multiple characteristic values, so the operations are queued and at the end, a final packet to commit the pending write operations and execute them is sent.

Write Without Response is the equivalent to notifications, except that here the data stream is from client to server. This uses Write Command packets, which are not acknowledged at the application layer, only by the link layer, which means that the client will get the acknowledgement that the packet was successfully transferred to the server, but not if the server accepted the packet. The data payload limit of $ATT\_MTU - 3$ bytes applies to *Write Without Response* as well.

The *Signed Write Without Response* uses the same mechanism as *Write Without Response*, with the difference being it is only used if the authenticated bit is enabled and the client and server share a bond and that the data payload is $ATT\_MTU - 15$ bytes.

Characteristic Value Notification

This procedure is used when a server is configured to notify a Characteristic Value to a client without expecting any acknowledgement that the notification was successfully received. Notifications can be configured using the Client Characteristic Configuration descriptor.

Characteristic Value Indication

This procedure is used when a server is configured to indicate a Characteristic Value to a client and expects an acknowledgement that the indication was successfully received. Indications can be configured using the Client Characteristic Configuration descriptor.

3.5.5 BLE Packet

A BLE packet containing application data and transferred over the air has the structure seen in Figure 3.4. The size of the Data field is dependent on the Bluetooth specification version. In Bluetooth 4.0 and 4.1, the maximum size of the Data field was 27 bytes (+4 bytes for the message integrity check). Bluetooth 4.2 brought the new feature called *Data Length Extension (DLE)*, which enabled the Data field size to be up to 251 bytes (+4 bytes for the message integrity check).

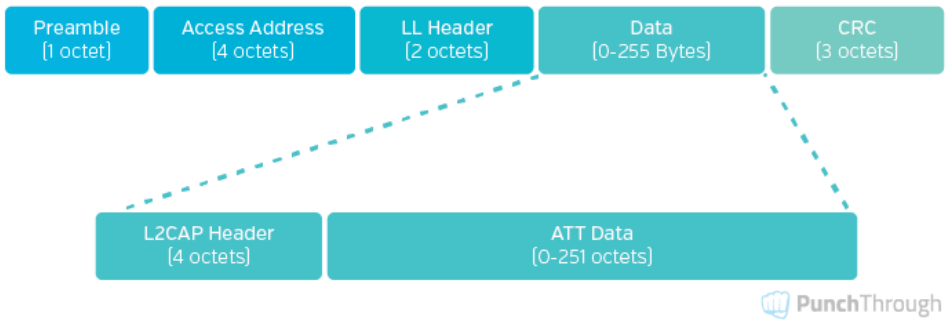

Figure 3.4: BLE packet structure[4].

That Data field is an L2CAP packet, which includes 4 bytes for the L2CAP header, yielding 23 (BLE v4.0, v4.1) or 247 (BLE v4.2, v5) bytes of ATT Data, which is also known as *ATT_MTU*. The ATT packet includes 1 byte for the ATT operation code and another 2 bytes for the attribute

[4]Image via PunchThrough: `https://punchthrough.com/blog/posts/maximizing-ble-throughput-part-2-use-larger-att-mtu`

Figure 3.5: ATT packet structure[5].

handle, as seen in Figure 3.5, from which we get the maximum data payload for the application layer of 20 $(27 - 4 - 1 - 2)$ bytes in BLE v4.0 and v4.1, and 244 $(251 - 4 - 1 - 2)$ bytes for BLE v4.2 onwards.

[5]Image via PunchThrough: `https://punchthrough.com/blog/posts/maximizing-ble-throughput-part-2-use-larger-att-mtu`

Chapter 4

Testing methodology

4.1 Hardware

Testing was done using two Nordic *nRF52840 DK* [6] development kit boards, one as the central device and the other as the peripheral. The *nRF52840 DK* is a single board development kit for Bluetooth development on *nRF52840 SoC* [21], with support for the on-board SEGGER J-Link OB Program/Debug probe [22]. The *nRF52840 SoC* is designed around an ARM Cortex-M4 CPU and has a 1MB flash with cache and 256kB RAM. It has the full hardware and software support for all the new Bluetooth 5 features: Long Range (Coded) and High-Speed PHY layers, Advertising extensions and improved coexistence. The Nordic protocol stacks are known as *SoftDevices* [23]. The nRF52840 is supported by the S140 SoftDevice, which is a 20-link Bluetooth 5 pre-qualified Bluetooth Low Energy protocol stack.

The central device was the test initiator and controller and was connected to a PC for test monitoring using the on-board SEGGER J-Link Real Time Transfer technology [24]. The peripheral device was the device under test and was configured in *nRF only mode*, where the *nRF52840 SoC* under test is decoupled from peripheral devices such as LEDs, interface MCU and external memory using analogue switches. It was then powered with 3.000V using the *external supply* connector on the board from the measurement system.

Range	Resolution	Accuracy [1]	Offset
1-70 µA	0.2 µA	±20%	±2µA
70 µA-1 mA	3 µA	±15%	±2µA
1-70 mA	50 µA	±15%	±2µA

Table 4.1: Power Profiler measurement resolution and accuracy.

Bluetooth Low Energy is backwards compatible down to version 4.0, which is why a single chip (*nRF52840 SoC*) was used for all the tests. Tests for versions BLE v4.0/v4.1 and BLE v4.2 were done with the same hardware as tests for BLE 5, with parameters set to that specific version capabilities, as described in Chapter 4.4.

4.2 Measuring system

For measurements, the *Power Profiler Kit* [7] from Nordic Semiconductor was used. The Power Profiler Kit is a current measurement tool for embedded development, specifically optimised for measuring the power consumption of Bluetooth devices. It has a three-stage analogue measurement circuit, which splits the total current range into three ranges: 1 µA to 70 µA, 1 µA to 1 mA and 1 mA to 70 mA. It can be used to power external boards that are being measured, with Vcc level configurable between 1.8 V and 3.6 V with up to 70 mA.

Table 4.1 shows the measurement resolution and accuracy of the Power Profiler Kit. It should also be noted that the measurement frequency is 77 kHz, which nets the time resolution of 13 µs and that the maximum recording / averaging time in the software is 120 s. Although we can see that the accuracy of the Power Profiler Kit is not the greatest, it is sufficient for our needs. That is because of the nature of the tests and the fact that they were all conducted in the same environment and under the same conditions.

[1] Accuracy of average readout

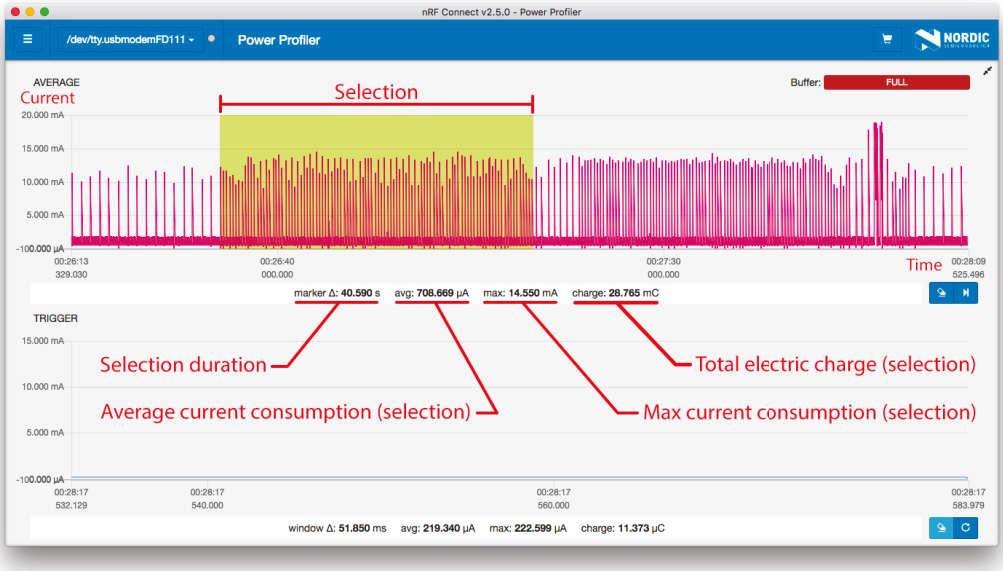

Figure 4.1: Power Profiler software with test data selected.

No drift of the baseline (radio off) consumption was detected during testing, which means that comparisons can still be made. Tests were also conducted multiple times and averaged to get the results.

The interface MCU of the child development kit is used for data connection to the Power Profiler Kit so that the data can be displayed in the PC software. The measurement data is displayed in dual measurement windows: one with longer, averaged acquisition times and one with high time resolution to show trigger events (if configured). A subsection of data can be selected for average current and energy calculations, as seen in Figure 4.1.

4.3 Hardware configuration

As explained in Chapter 4.2 a nRF52840 DK board is needed for Power Profiler Kit operation, which can also measure the consumption of the nRF52840 SoC on the DK board. It has been decided to separate the measurement sys-

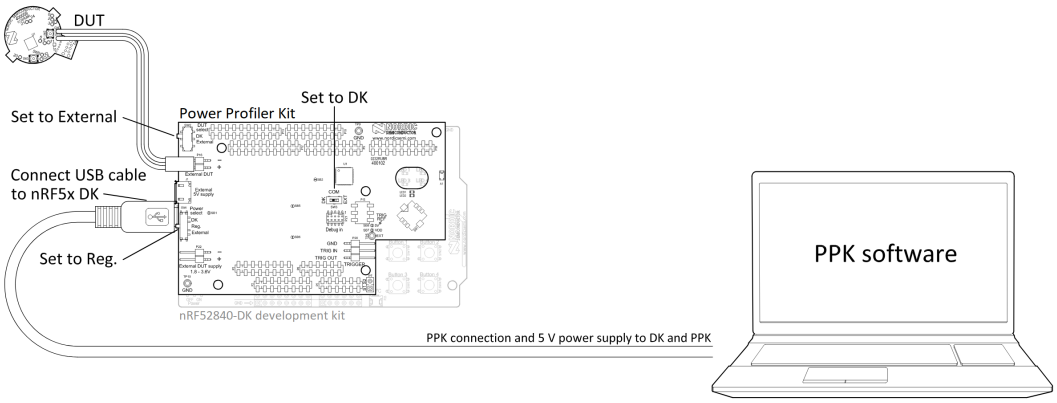

Figure 4.2: Power Profiler connected to the peripheral device under test (DUT) and a computer, running Power Profiler Kit Software[2].

tem and the device under test. This was done to avoid any interference that the Power Profiler Kit and the interface MCU might present to the current measurement, as well as the Bluetooth connection. The connection can be seen in Figure 4.2, with the DUT being our peripheral device, connected to the external supply connector.

The actual hardware connections can be seen in Figure 4.3, where a small resistor can be observed between pin 13 and ground. The resistor is 470 Ω 1%, measured at 473 Ω. When the voltage is applied to pin 13, this resistor will increase the constant consumption by ~5.5 mA, as per the Ohm's Law:

$$I = \frac{V}{R} = \frac{V_{OH}}{R} = \frac{V_{dd} - 0.4\,\mathrm{V}}{R} = \frac{3.0\,\mathrm{V} - 0.4\,\mathrm{V}}{473\,\Omega} = 5.5\,\mathrm{mA} \tag{4.1}$$

, where V_{OH} is output high voltage, which is equal to $V_{dd} - 0.4\,\mathrm{V}$ [21]. This is used as an anchoring mean when conducting tests to roughly mark the start and end of the test. While the peripheral device is waiting for commands, the pin 13 is kept high, allowing the current to flow through the resistor and thus increasing the consumption. Once we get the *test start* command, pin 13 is

[2]Image via Nordic: `http://infocenter.nordicsemi.com/topic/com.nordic.infocenter.tools/dita/tools/power_profiler_kit/PPK_user_guide_PPK_on_customHW_with_nRF5xDK.html?cp=5_6_5_4`

kept low until the test is done, to not interfere with the test measurements, but still generate an edge, visible on the measurement software. An example of how the edge looks in the measurement software can be seen in Figure 6.1, on the very left side of the measurements, where the consumption falls from ~5 mA to ~0.6 mA, and at the right side where it rises again. The rise was configured with a delay of one connection interval, to insure separation of test measurements.

Figure 4.4 shows the devices while testing and measurements are undergoing. The peripheral device under test is flipped on top of the central device. That is done to minimise the effect of interference on tests, as the signal to noise ratio will be best if devices are close.

4.4 Test scenarios

It was decided not to test the BLE 5 Long Range, using 125 or 500 kbps Coded PHY, as the increased range offered can't be achieved without it, which would justify lower throughput and consecutively higher consumption for the transmitted amount of data. Because of the much-increased range, it also makes no sense to compare it to uncoded 1 Mb/s and 2 Mb/s PHYs, which can not even connect at the extended range that coded PHYs are offering.

Before doing the comparison tests, some theoretical calculations were done in regards to the expected time needed to transfer data with larger connection intervals (CIs), because of the 120 s recording limit of the measurement software. This was especially important for read and write operations, as they take two connection events (CE) to complete. In the first CE, the read/write command is sent to the peripheral and in the second CE the reply is sent to the central device.

$$packets = \frac{payload}{ATT\_MTU - 3} \tag{4.2}$$

$$time = packets * CI * 2 \tag{4.3}$$

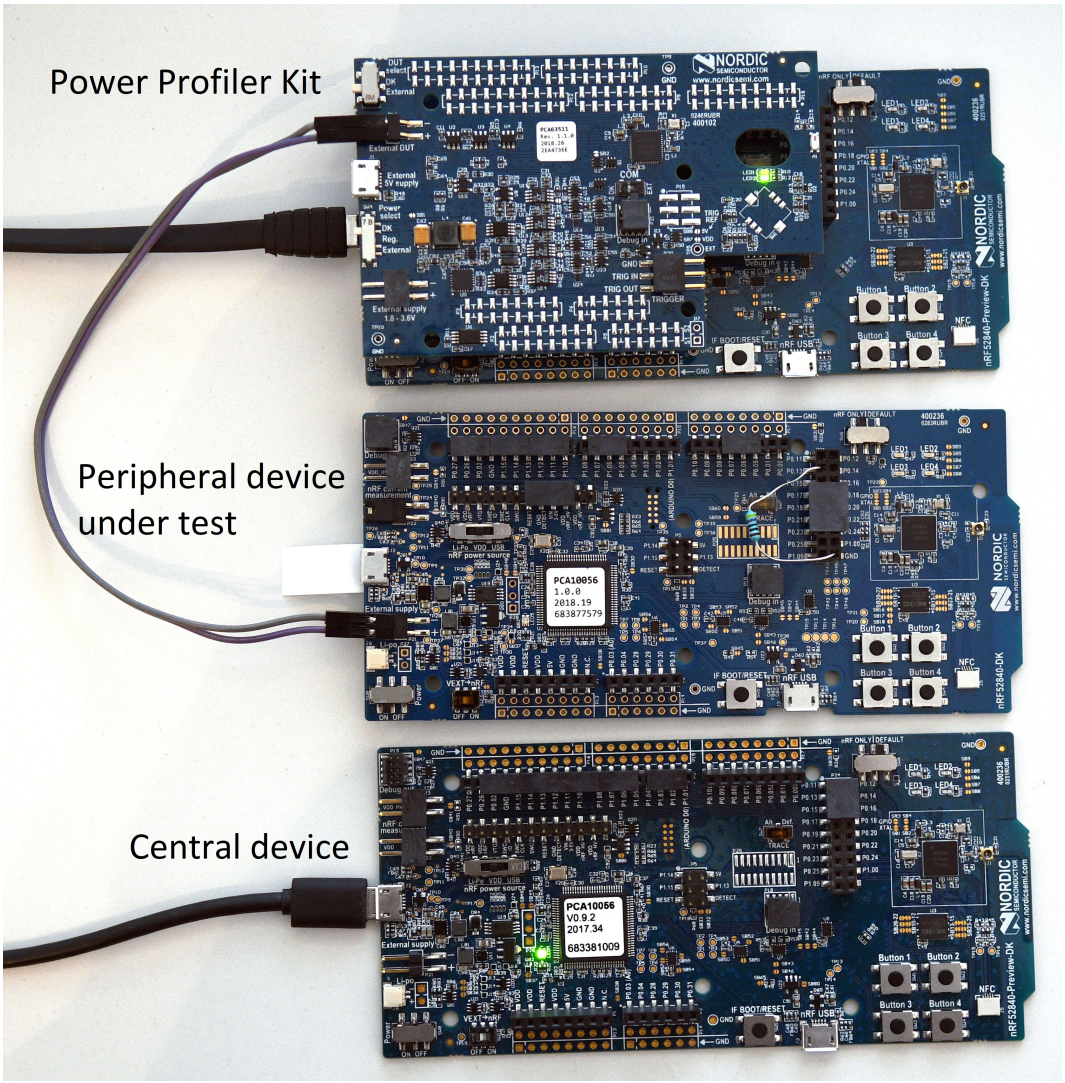

Figure 4.3: Hardware connections. Top board: Power Profiler on nRF5240 DK, the middle board: peripheral device under test, the bottom board: central device.

From Equations 4.2 and 4.3 we have generated Tables 4.2 and 4.3. Here it can be seen that with bigger payloads at long connection intervals, the time that a single test would take is larger than the measuring time limit of 120 s that the measuring system used. To overcome this barrier, tests were

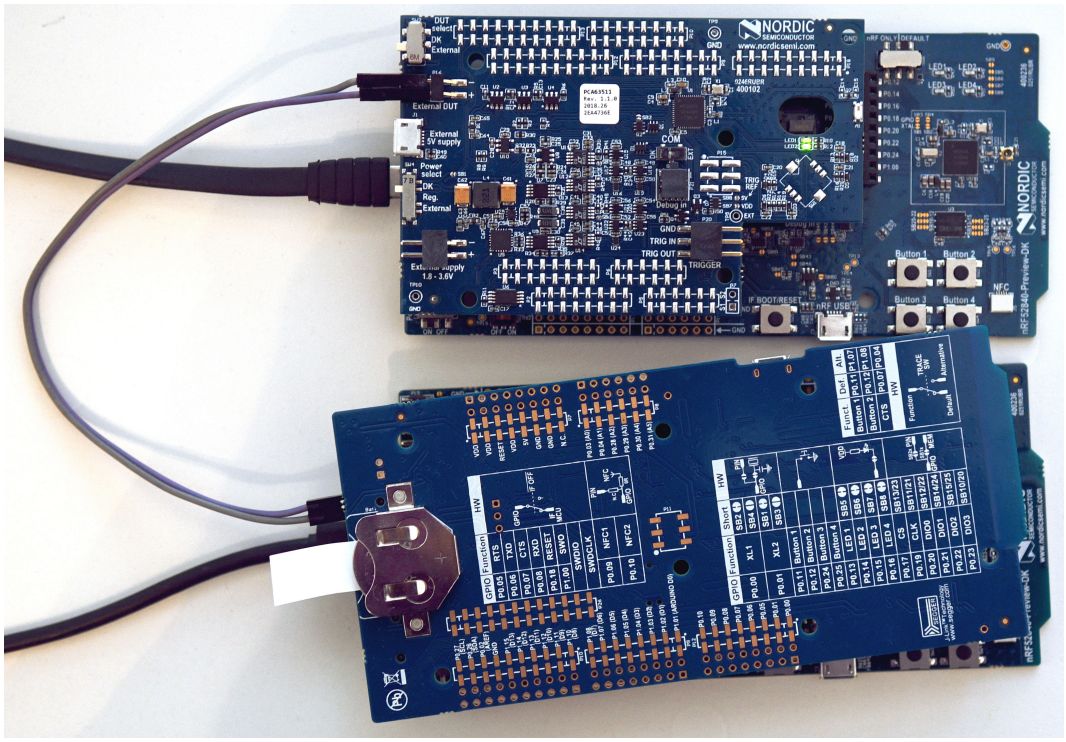

Figure 4.4: Hardware connections during testing. Top boards: Power Profiler on nRF5240 DK, bottom boards: peripheral device under test flipped on top of the central device.

done that are designed to test invariance of average power consumption from data payload above a certain limit at different connection events. The test parameters for these tests can be observed in Table 4.2, where the maximum times, that we could measure energy consumption, using our measurement system, are highlighted in green.

As seen in Table 4.3, this is much less of a problem with BLE v4.2 and BLE 5, as they allow a maximum payload of 244 bytes ($ATT\_MTU - 3$), which decreases the number of packets needed for data transmission and thus the time needed for this transmission by a factor of ~10, as we increase payload size.

		Time [s] @ connection interval				
Payload	Packets	@ 7.5ms	@ 50ms	@ 400ms	@ 1000ms	@ 4000ms
20 B	1	0.015	0.1	0.8	2	8
100 B	5	0.075	0.5	4	10	40
400 B	20	0.3	2	16	40	160
1 KB	50	0.75	5	40	100	400
10 KB	500	7.5	50	400	1000	4000
100 KB	5000	75	500	4000	10000	40000

Table 4.2: Packets and time needed to transfer different payloads in relation to connection intervals for BLE v4.0 / v4.1 with *ATT_MTU* set to 23, using read/write operation. Maximum times that could be measured with our equipment are highlighted in green.

		Time [s] @ connection interval				
Payload	Packets	@ 7.5ms	@ 50ms	@ 400ms	@ 1000ms	@ 4000ms
20 B	1	0.015	0.1	0.8	2	8
100 B	1	0.015	0.1	0.8	2	8
400 B	2	0.03	0.2	1.6	4	16
1 KB	5	0.075	0.5	4	10	40
10 KB	41	0.615	4.1	32.8	82	328
12.5 KB	52	0.78	5.2	41.6	104	416
100 KB	410	6.15	41	328	820	3280

Table 4.3: Packets and time needed to transfer different payloads in relation to connection intervals for BLE v4.2 with *ATT_MTU* set to 247, using read-/write operation.Maximum times that could be measured with our equipment are highlighted in green.

In general, tests with small data payloads were not carried out, as with small payloads (smaller than *ATT_MTU* for example) the benefits of newer and faster Bluetooth standards do not come in to play. Tests of medium data payloads (a couple of *ATT_MTU*s) were omitted as well, because the differences between Bluetooth versions would be smaller, and we could not reap the full benefits of Bluetooth 5. Tests for comparison between standards were done with large data payloads, as seen in Table 4.4. The improvements that BLE 4.1 brought over BLE 4.0 were virtually non-existent as far as hardware, throughput, range and power consumption go. Because of that, tests for BLE v4.0 and BLE v4.1 were bundled together and done by using exactly the same parameters.

BLE version	Constant Parameters	Connection Intervals [ms]	Procedure	Payload
v4.0/v4.1	*ATT_MTU*: 23 TX power: 8 dBm PHY: 1 Mbps Distance: 2cm	7.5, 30, 75, 150, 400, 1000	Read, Write	1 KB
			Notify Write w/o rsp	100 KB
v4.2	*ATT_MTU*: 247 TX power: 8 dBm PHY: 1 Mbps Distance: 2cm	7.5, 30, 75, 150, 400, 1000	Read, Write	12.5 KB
			Notify Write w/o rsp	100 KB
v5	*ATT_MTU*: 247 TX power: 8 dBm PHY: 2 Mbps Distance: 2cm	7.5, 30, 75, 150, 400, 1000	Read, Write	12.5 KB
			Notify Write w/o rsp	100 KB

Table 4.4: Test parameters for testing the energy efficiency of different Bluetooth standards.

4.5 Measurement strategy

All measurements were conducted five times, to achieve some degree of confidence, as well as to try to improve on the measuring system accuracy. The measured values were averaged using the arithmetic mean method.

All measurements were performed over the complete transaction. End of the transaction was defined as the point in time when the next transaction of the same type can be conducted on the same characteristic. This means that for GATT read and write operations, the measurements were taken over 2 times the number of data packets connection intervals (as presented in Equation 4.3). This is due to the fact that each data packet write takes up two connection intervals: the first one for data packet transmission and the second for GATT write acknowledgment. The following packet transmission can be made in the next (third in this example) connection event, as each GATT write request needs its GATT write reply before the transaction is complete. Notifications and writes without response, on the other hand, do not need any GATT level replies and can be chained together. Because of that, measurements of notify and write without response operations were stopped when all the data was transferred, even when data transfer was completed in the middle of the connection interval, as the next transaction can be started immediately after that. The measurement intervals and measured quantities are summarised in Table 4.5.

Operation	Measurement interval	Measurements
Read	From the first GATT request to the end	Time
Write	of connection interval of the last GATT reply	Average current
Notify	From start of first GATT request to	Energy consumption
Write w/o rsp	end of last GATT request	Power consumption

Table 4.5: Parameters of measurements performed on test cases described in Chapter 4.4

The time taken to transfer the data was reported by the custom firmware

running on Bluetooth devices. In the Power Profiler software, that time was selected and visually confirmed, as it is described in the Chapter 6.1. Once the time window was set, the software calculated the total charge consumed and average current in the defined time window. Details of implementation of these calculations in the measurement software are not known.

We recorded the duration of the time window, which was marked by the firmware, and the average current in the time window read from the Power Profiler software in a spreadsheet. Those two measurements, combined with the test scenarios parameters described in Chapter 4.4 and constant power supply voltage of 3.00 V were used to calculate throughput using Equation 4.4, average power consumption using Equation 4.5 and energy consumption using Equation 4.6, for each measurement.

$$throughput = \frac{data\ size}{t} \tag{4.4}$$

$$P = UI \tag{4.5}$$

$$E = Pt \tag{4.6}$$

Measurements were averaged using the mean method with Equation 4.7 and standard deviation was calculated using Equation 4.8 for all test cases described in Chapter 4.4. All measurements and calculated results are published on GitHub [25].

$$\bar{x} = \frac{\sum_{i=1}^{N} x_i}{N} \tag{4.7}$$

where:

$\bar{x}$ = arithmetic mean
N = the number of terms (e.g., the number of items or numbers being averaged)
x_i = the value of each individual item in the list of numbers being averaged

$$\sigma = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \bar{x})^2}{N}} \tag{4.8}$$

where:

σ = standard deviation

N = the number of terms (e.g., the number of items or numbers being averaged)

x_i = the value of each individual item in the list of numbers being averaged

$\bar{x}$ = arithmetic mean

Chapter 5

Firmware implementation

A highly modular system was built from scratch, specifically for the purpose of these tests. The projects for the central and the peripheral device are separated, but they re-used a lot of lower layer abstraction layers, which have been built on top of *Nordic BLE Stack Components* to provide an even higher abstraction to the topmost layers. The abstraction stack architecture can be observed in Figure 5.1. A timer was needed on both the central and peripheral device for the purpose of conducting these tests.

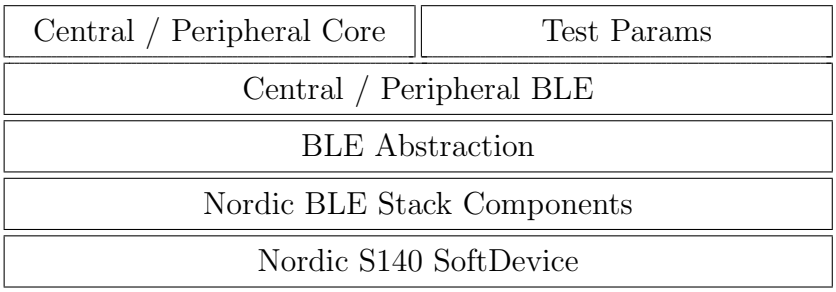

Figure 5.1: Firmware abstraction stack architecture.

The layers and their functions are:

- **Central / Peripheral Core** is the module that contains the testing logic. This is implemented using a state machine, which initialises the device on boot and awaits for further instructions.

- **Test Params** is the module, shared between central and peripheral devices by the use of *git submodule* mechanism [26] to keep it up to date in both projects. It is used to load the test parameters for different Bluetooth versions, as well as data building and checking.

- **Central / Peripheral BLE** is the module that maps the internal handles to UUIDs, as well as forwarding relevant Bluetooth events to the Core layer. On the central device, it is also responsible for scanning, connecting and service/characteristic discovery procedures.

- **BLE Abstraction** provides a layer of abstraction over the different Nordic BLE Stack Components.

- **Nordic BLE Stack Components** are the Host part components abstractions of the BLE Stack, provided by Nordic Semiconductor. These include GATT, GAP, L2CAP and other BLE layers and modules, as well as hardware abstraction layer.

- **Nordic S140 SoftDevice** is a pre-compiled binary image and functionally verified according to the wireless protocol specification, revealing only its Application Programming Interface (API), which is abstracting the actual procedures within.

5.1 Peripheral Core

The *Peripheral Core* is used on the peripheral device as the main decision-making module. It implements a state machine with the following states:

- The **Initialisation state** is the state which performs the initialisation of the whole device on boot. This includes initialising the BLE stack, GAP and GATT layers, timers, initial connection parameters, services and characteristics, the *Peripheral BLE* module and starts the BLE advertising procedure.

- The **Idle state** is the state after the initialisation and after each test, where the peripheral device waits for commands from the central device.
- The **Test start state** is the state which the device enters after it receives the test start command. Here the test parameters and variables are initialised.
- The **Test run state** is the state in which the peripheral device is during a test is running. It checks for test completion or termination command and in the case of the test using the notifications, it puts the notification operations into the queue, until the test is complete.
- The **Notification self-test state** is a state which is entered when the Notifications self-test command is received. This is not a part of the tests, it is only used to check if notifications work after the central device connects. It issues a single notification.
- The **Wait for notifications done state** is where the peripheral device waits for a queued notification to be done. This is used to avoid a notification queue overflow while allowing for normal operation of all the lower layers.
- The **Test terminate state** is entered after test terminate command is received. This state terminates the current running test and resets all the test variables.
- The **Delay state** is used to delay the switch to a different state, while allowing for the rest of the layers to continue working.

The state machine is implemented in the peripheral_core_update() function, which is continuously called from the main() function. After initialisation, it starts the advertising procedure and waits for the central device to connect to it. Once the central device is connected, the central device delegates all the following operations by the use of the control characteristic. The core

module listens for events generated by the *Peripheral BLE* layer (Chapter 5.2), such as Connection and Disconnection events, Notification subscription event, Write events to control and data characteristics, Data characteristic read reply sent event and Data characteristic notification sent event. The control characteristic writes are parsed for commands and actions are taken depending on the command issued, while writes to Data characteristic are checked for data correctness with the help of Test Params layer described in Chapter 5.5.

5.2 Peripheral BLE

The *Peripheral BLE* layer provides a level of abstraction to the *Peripheral Core* layer. It houses functions that abstract the peripheral initialisation of services and characteristics (function calls to *BLE Abstraction* layer), sending notifications (function calls to *BLE Abstraction* layer) and functions that abstract events generated by lower layers (*Nordic BLE Stack Components*. The lower layers call each specific event handler function implemented in the *Peripheral BLE* module, which processes the event and calls the *Peripheral Core* event handler with the event ID and the event data.

5.3 BLE Abstraction

The *BLE Abstraction* layer provides and additional layer of abstraction to the *Peripheral BLE* layer for initialising services and characteristics, sending notifications, and tying the internal characteristic structs to the respective UUIDs.

5.4 Central Core

The *Central Core* is used on the central deviceas the main decision making module. It implements a state machine with the following states:

- The **Initialisation state** is the state which performs the initialisation of the whole device on power up. This includes initialising the BLE stack, GAP and GATT layers, timers, initial connection parameters, service and characteristic discovery module, the *Central BLE* module and starts scanning for the peripheral device.

- The **Idle state** is the state after the initialisation and after each test, where the centra device waits for the connection event or the test start button press event when the connection is made.

- The **Write self-test state** is entered after the connection has been made and service / characteristic discovery is complete. It issues a single write to the Control characteristic, to test if writes work.

- The **Read self-test state** is entered after the write self-test is complete. It issues a single read to the Control characteristic, to test if reads work.

- The **Test init state** is entered when a test is started. In this state, the parameters defined for the current test are set.

- The **Test wait for parameters state** is entered right after the *Test init state*. The central core uses this state to confirm all the test parameters are set before continuing.

- The **Test init 2 state** is entered after test parameters are confirmed. The test parameters are serialized and sent to the peripheral device using the Control characteristic.

- The **Test start state** is entered after the test parameters have been sent to the peripheral device. The *Test start* command is sent to the peripheral device. This marks the start of the test.

- The **Test run state** is the state in which the central device is during a test is running. It uses a switch case to differentiate procedures used

to perform different tests (write, read, notify, write without response). If the current test is testing with notifications, nothing is done on the central device once the test is started, as the notifications are performed from the peripheral device, so the central device only waits for test completion.

- The **Test complete state** is entered after a test was complete. It calculates the time needed for the test to run and average throughput of the completed test and resets the current test variables.

- The **Test terminate state** is entered if the button to terminate the current state is pressed. This state terminates the current running test and resets all the test variables, as well as sends the terminate test command to the peripheral device.

- The **Wait for read done state** is where the central device waits for a read to be completed.

- The **Wait for write done state** is where the central device waits for a write or write without response to be completed.

- The **Delay state** is used to delay the switch to a different state, while allowing for the rest of the layers to continue working.

The state machine is implemented in the central_core_update() function, which is continuously called from the main() function. After initialisation, it starts the scanning procedure, waits for the connection to be made and service / characteristic discovery to finish, after which it tests reads, writes and waits for a notification from the peripheral. After the self-tests are finished, it waits for the user to start the tests by pressing one of the hardware buttons. On button press, the specified tests are queued up and ran one after the other. Before the start of each test, the test parameters are applied, after which the test parameters are transmitted to the peripheral via control characteristic. Once all the parameters are confirmed by both sides, the *start test* command

is sent to the control characteristic. The actual test is performed over the data characteristic. Received data is checked and the test is terminated if the data is incorrect at any point. The core module listens for events generated by the *Central BLE* layer (Chapter 5.6), such as connection and disconnection events, service and characteristic discovery done event, read, write and write without response events, notification received events, connection parameters updated events and PHY layer updated events. A listener for button press events is implemented, which queues different tests and starts the first one on button press.

5.5 Test params

The *Test params* module is where the parameters for the tests are described. A custom struct is defined as seen on Listing 5.1, which encapsulates the test parameters, so that they can be passed around and used by the *Central core* module.

```
typedef struct
{
  test_case_t test_case;
  uint16_t att_mtu; // GATT ATT MTU, in bytes.
  float conn_interval; // Connection interval in ms
  bool data_len_ext_enabled; // Data length extension status.
  bool conn_evt_len_ext_enabled;
     // Connection event length extension status.
  uint8_t rxtx_phy; // PHY used (125 Kbps, 1 Mbps or 2 Mbps)
  int8_t tx_power; // Output power
  uint8_t link_budget;
     // Link budget (output power minus sensitivity)
  uint32_t transfer_data_size; // Transfer data size in bytes
  test_ble_version_t ble_version;
} test_params_t;
```

Listing 5.1: Custom struct that encapsulates the test parameters

The *Test params* module houses some abstractions for setting some of

the test parameters, such as setting the *ATT_MTU*, data length extension, as well as a function that set all the parameters, as they are set in the input test_params_t struct. There is a function that loads the test parameters to a test_params_t struct, depending on the BLE version selected, as well as test parameters serialization and deserialization helper functions. There are also functions to build data and confirm incoming data. Data is always built as an increasing offset of the current byte in the running test, of which the lowest 8 bits are taken. This way, all the possible bytes are sent repeatedly in a single test, negating potential effects of static data on test results.

5.6 Central BLE

The *Central BLE* layer provides a level of abstraction to the *Central Core* layer. It abstracts operations such as initialisation of the discovery module, enabling the notifications, reading, writing and writing without response, setting connection parameters and functions that abstract events generated by lower layers (*Nordic BLE Stack Components*. The lower layers call each specific event handler function implemented in the *Central BLE* module, which processes the event and calls the *Central Core* event handler with the event ID and the event data.

5.7 Nordic SoftDevice

The Nordic SDK version used was 13.1.0, with SoftDevice S140 v5.0.0-2.alpha, as that was the latest SDK and SoftDevice combo that Nordic offered at the start of programming. Because SDK version 14.x.x was still using the same SoftDevice and did not offer any improvements related to testing I decided not to migrate to the newer SDK. With SDK version 15.0.0 came SoftDevice S140 v6.0.0, which is production tested for nRF52840 SoC and brought support for LE Coded PHY S=2 (500 kbps). At that point, the migration would have taken too much time, and the implementation of LE

Coded PHY S=2 was of no direct interest for these tests, so it has been decided against it. The same reason applies to not migrating to SDK v15.2.0 and SoftDevice S140 v6.1.x.

5.8 Bluetooth Low Energy API

The peripheral device implemented one proprietary Service, with two proprietary Characteristics, as seen in Figure 5.2. The test control and test data streams were split to ensure better control over tests. The first byte of the control characteristic was the control command enumeration, while the rest of the bytes were command dependent.

Test Service UUID: F0018000-F001-F001-F001-F001F001F001	
Test control characteristic UUID: F0010000-F001-F001-F001-F001F001F001	
Value	read, write
Test data characteristic UUID: F0010001-F001-F001-F001-F001F001F001	
Value	read, write, notify
Descriptor: CCCD	read, write

Figure 5.2: Peripheral device BLE API.

The Test control characteristic is used to control the tests. The control commands implemented are: Write test parameters, test start, test pause and test terminate, as well as the notification self-test command. The test data is transmitted over the Test data characteristic.

Chapter 6

Results and analysis

6.1 Measurement examples

Power consumption measurement for different BLE standards and different BLE operations were conducted. The power consumptions of operations with and without GATT acknowledgement were made using a different set of connection parameters (e.g. connection interval).

The Power Profiler software shows a scale starting at $-100\,\mu A$ by default, to be able to draw the signal line thickness, even if the signals are always above $0\,\mu A$. This may make it seem as if the signal is negative, as it can be observed on Figure 6.1, but if we were to change the scale, it becomes apparent that the signal is in fact positive but close to 0, as it can be observed on Figure 6.4.

First, power consumption measurements for read and write operations with GATT acknowledgement were made. An example of measurement of power consumption of the BLE 4.0 or 4.1 device for this operation (mode) is shown in Figure 6.1. The separate packets are clearly visible, as well as radio activity at the start of each connection event. It can also be observed when the radio activates, when data was actually transmitted, since the device consumed a bit more energy than when the GATT write with the data was received. It can also be seen how the device turned off the additional shunt

resistor just before the test started and turned it back on just after the test finished, with a delay of one connection event, so that it doesn't skew our measurements. Keep in mind that Link Layer reply is sent for each packet to ensure reliability. As predicted by calculations shown in Table 4.2, five packets were needed to transfer 100 bytes of data, which occupied ten connection events and took 75 ms to complete. The GATT level requests (red) and replies (green) are annotated on Figure 6.1. We can see ten connection intervals highlighted, as this is an example of GATT write of 100 bytes of data. The 100 bytes get split into five GATT write packets, each of which needs its GATT level reply, which nets the ten connection intervals occupied by this transaction.

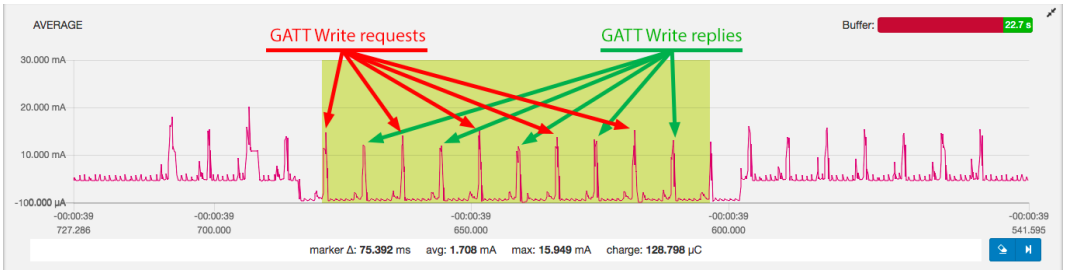

Figure 6.1: Measurement example: BLE 4.0/4.1 (*ATT_MTU*: 23), write 100 bytes, connection interval 7.5ms. The Power Profiler software shows a scale starting at −100 µA by default, to be able to draw the signal line thickness, even if the signals are always above 0 µA.

An example of power consumption measurements for BLE 4.2, with *ATT_MTU* set to 247 bytes for the reading of 12.5 KB of payload data, using connection interval of 75 ms, is depicted in Figure 6.2. Even though each packet can transfer roughly 10x more data than with BLE 4.1 because of the increased *ATT_MTU*, the throughput is still bound by the need for GATT replies, which for GATT read operation come in the next connection interval, meaning that a single GATT read operation occupies two connection events.

Lastly, measurement example for BLE 5 can be seen in Figure 6.3, where *ATT_MTU* is set to 247 and 2 Mbps PHY is used for transferring 100 KB of

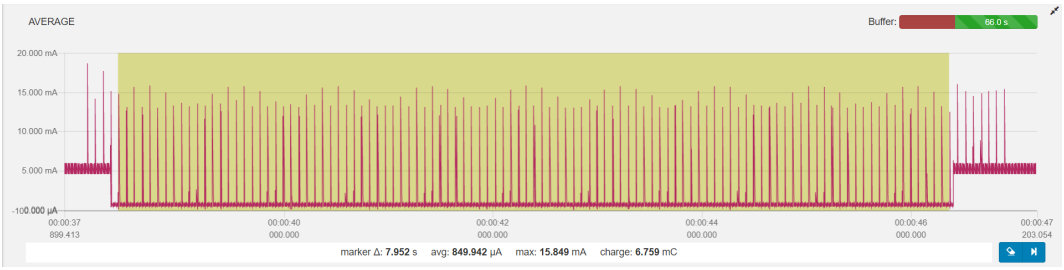

Figure 6.2: Measurement example: BLE 4.2 (*ATT_MTU*: 247), read 12.5 KB, connection interval 75ms.

data using notifications, with the connection interval set to 150 ms. Because notify operations (opposed to read and write operations) do not need the GATT layer replies but rely solely on Link Layer replies, multiple packets can be sent in the single connection event. We can see that in this instance, data transmission took just shy of four full connection events. At the start of each connection event, the central device still needs to start the connection event, as well as allow a bit of space for other possible communications to occur.

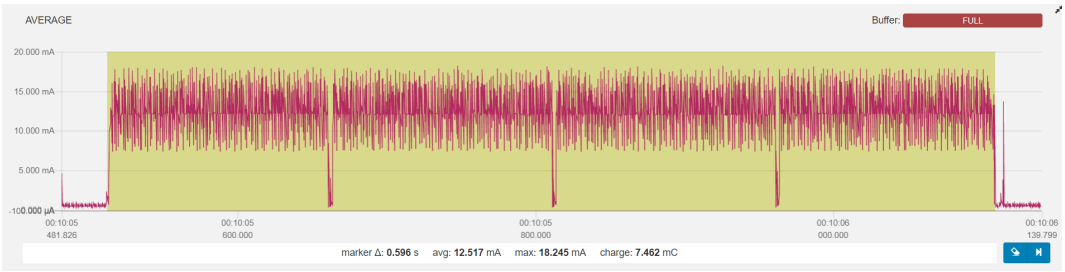

Figure 6.3: Measurement example: BLE 5 (*ATT_MTU*: 247), notify 100 KB, connection interval 150ms.

Measurement examples of the peripheral device in it's idle state when Bluetooth radio is turned off can be seen on Figure 6.4 and Figure 6.5. These measurements were done to measure the idle power consumption of the peripheral device, with and without the timer running, as this idle con-

sumption has an impact on overall consumption during the tests, particularly with larger connection intervals, where idle consumption can overshadow differences in BLE consumption with different connection parameters. Because the timer was needed to be able to conduct our tests, we had relatively high idle current consumption, which had an impact on our measurements. The idle current consumption with the timer on was measured to be 588.49 µA and with the timer off 7.86 µA, which means that the timer was using roughly 580 µA.

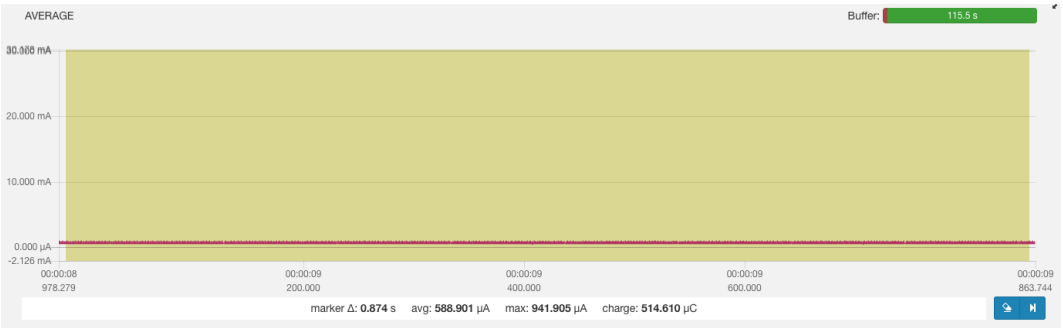

Figure 6.4: Measurement example: Peripheral device in idle state with BLE inactive and timer active.

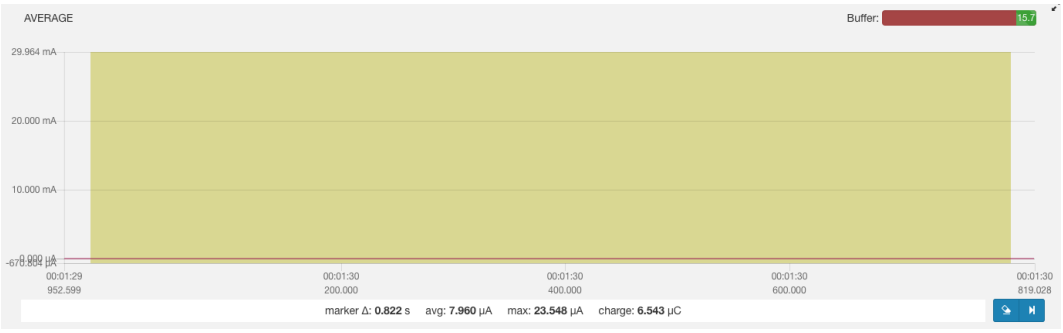

Figure 6.5: Measurement example: Peripheral device in idle state with BLE inactive timer off.

6.2 BLE 4.0/4.1 energy consumption

Energy consumption of the Bluetooth v4.0 and v4.1 device can be observed in Figures 6.6 for writing and 6.7 for reading. Here we can observe that energy consumption has linear dependence on the total transferred amount of data. This falls in line with theoretical expectations, as a single packet is transferred every two connection events (the second connection event is used for GATT level reply) and the energy consumption for a single packet in a set connection event is constant. Larger data sizes simply mean that more packets are sent, which in turn increases the energy consumption by the extra amount of data sent. It is because of this linear dependence that we were able to use different data sizes for BLE 4.0 / 4.1 tests than for BLE 4.2 and BLE 5. BLE 4.2 and BLE 5 use a larger ATT_MTU, so the test data size needed to be increased, in order to maintain roughly the same amount of packets sent.

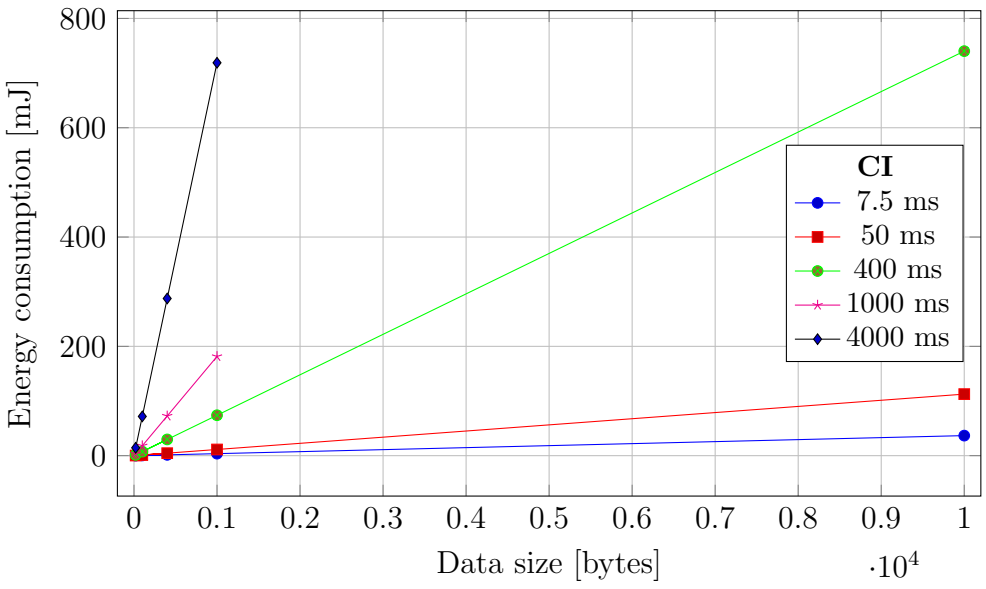

Figure 6.6: BLE 4.0/4.1 Energy consumption in relation to data payload size for write operation. Note: Measurements were not done for 10 kB data size for 1s and 4s connection intervals.

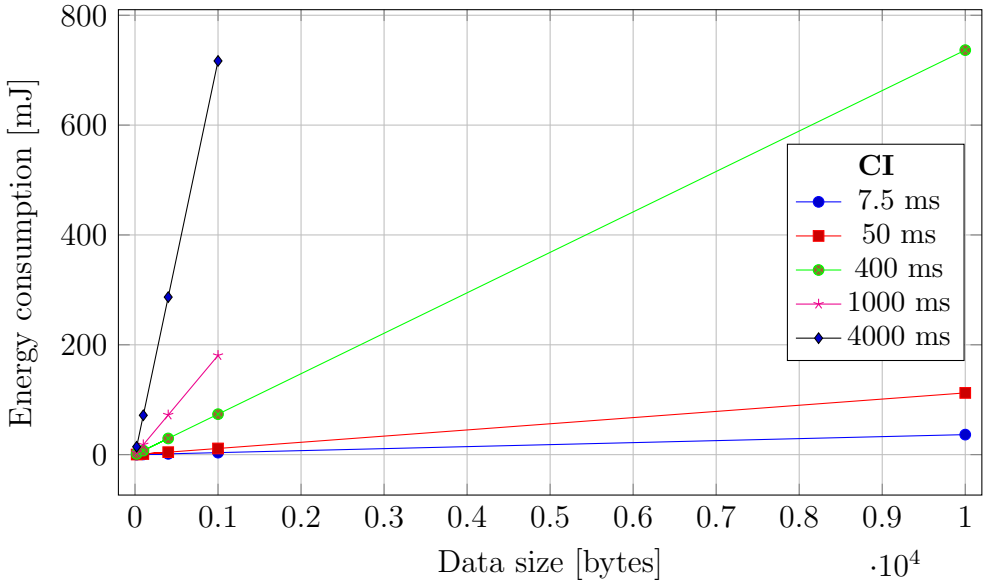

Figure 6.7: BLE 4.0/4.1 Energy consumption in relation to data payload size for read operation. Note: Measurements were not done for 10 kB data size for 1s and 4s connection intervals.

Usage of a timer, which is needed to conduct our tests, introduced an offset of roughly 580 µA into the idle current consumption. This means that at large connection intervals the idle energy consumption overshadowed the energy consumption of BLE operations. When we subtracted this offset, we got Figures 6.8 and 6.9. Here we can see that the linear dependence of energy consumption on data size still holds. The differences in energy consumption between different connection intervals when testing with the same data size are smaller, but they still exist because even without the timer there is some small idle device energy usage, which adds up over time, particularly with larger connection intervals.

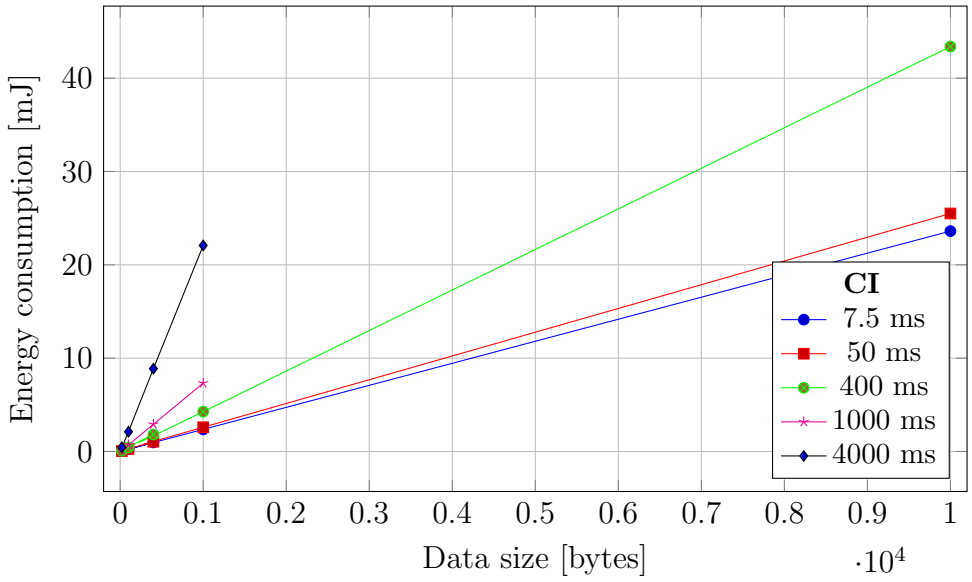

Figure 6.8: BLE 4.0/4.1 Energy consumption in relation to data payload size for write operation. Constant energy consumption by the on-board timer was subtracted from the data. Note: Measurements were not done for 10 kB data size for 1s and 4s connection intervals.

6.3 Throughput

The primary means of changing throughput in Bluetooth Low Energy are *ATT_MTU*, connection interval and PHY modulation speed. Since the parameters *ATT_MTU* and PHY determine the Bluetooth standard of the device, the only parameter needed to tune throughput is the Connection Interval.

The throughput of reading operation is limited by the fact that a GATT reply is needed, and it is sent in the connection event that follows the read command. The relationship between data throughput and connection interval is depicted in Figure 6.10. We can see that there is a clear correlation between connection interval and throughput. The reason for the correlation is that as we shorten the connection interval, the faster we get the reply, meaning we can send the next read command faster.

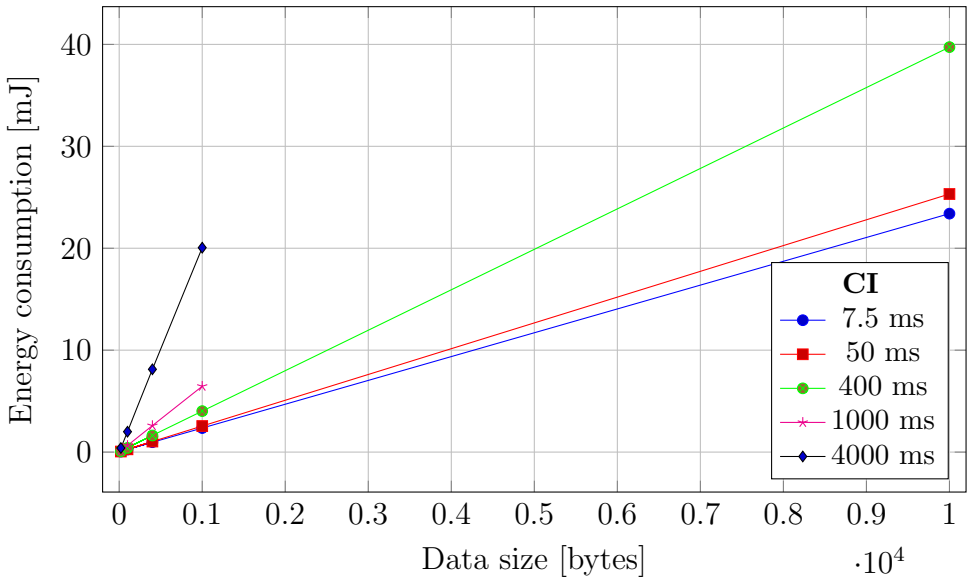

Figure 6.9: BLE 4.0/4.1 Energy consumption in relation to data payload size for read operation. Constant energy consumption by the on-board timer was subtracted from the data. Note: Measurements were not done for 10 kB data size for 1s and 4s connection intervals.

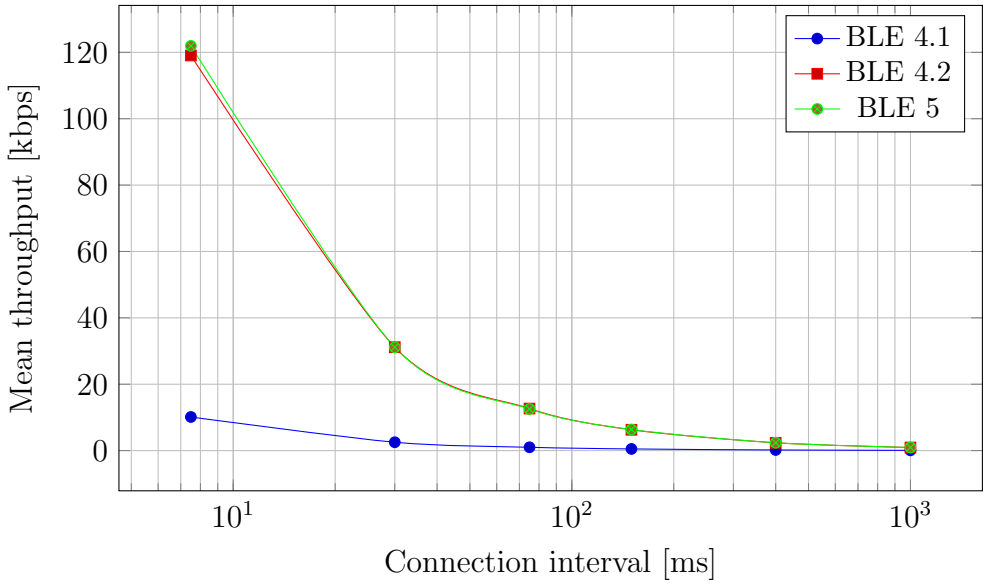

Figure 6.10: Mean read throughput of BLE versions at different connection intervals.

Notifications, on the other hand, are not limited by higher layer replies, as they rely solely on Link Layer acknowledgements, which are sent in the same connection event. This means that in theory, if we can transmit all the data in a single connection event, the highest throughput will be achieved. In practice, this is not true, as it can be seen in Figure 6.11. Starting from the shortest CI of 7.5 ms and going to the longest, we can see that at the beginning the throughput is increasing as we increase the connection interval. That is because there is a fixed amount of time required at the connection event when the data is not transmitted yet. With longer connection interval, the portion of the data transmission rises. Somewhat surprisingly, as we increase the connection interval further, throughput starts to decrease at some point. This happens in practice, as there is interference from other devices in the area, which causes some packets to fail in transmission. If a notification packet fails, the connection event ends and the devices wait for the start of the next connection event. This causes the decrease in throughput, which can be observed in Figure 6.11 after connection interval is increased

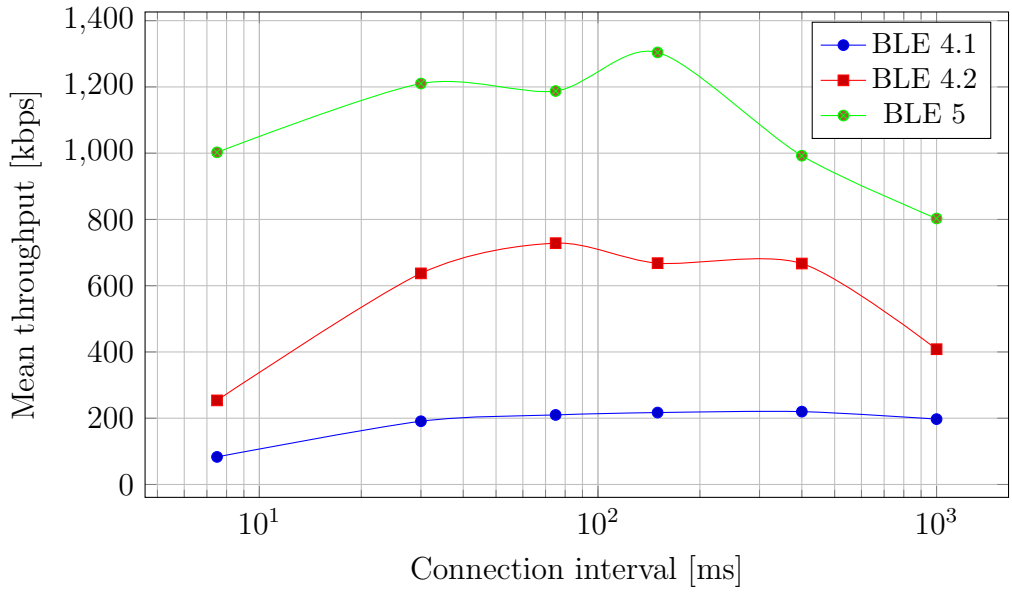

Figure 6.11: Mean notification throughput of BLE versions at different connection intervals.

sufficiently.

Theoretical maximum data throughput of different BLE standards was studied in [27]. The results are summarized in Table 6.1 where the modulation rate, packet time, and max throughput are given. Comparing this result with our measurement we can see that peak performance of each BLE standard is close to given bounds.

BLE version	Modulation rate	Packet time	Max Throughput
v4.0 / v4.1	1 Mb/s	708 µs	0.305 Mb/s
v4.2	1 Mb/s	2500 µs	0.803 Mb/s
5	2 Mb/s	1400 µs	1.4 Mb/s

Table 6.1: Maximum theorethical throughput and time to transmit a single packet by different BLE specification versions [27].

6.4 BLE version comparison: read and write

Read and write operations are bound by same limits of waiting for the response in the next connection event, so they also act the same under testing and gave similar results. This is the reason we have included a single graph for read and a single graph for write results. As shown in Figure 6.10 and explained in Chapter 6.3, for read and write, throughput is increased when connection interval is shortened and vice versa.

The relationship between power consumption and data throughput for different BLE standards is given in Figure 6.12. From looking at Figure 6.12 we can see the improvements through the BLE standard iterations, and it is interesting to see that BLE 5 is consistently using less power at same throughput than BLE 4.2. That is because even though the 2 Mbps PHY modulations needs more power, it also almost halves the time to send a single packet in comparison to BLE v4.2, as seen in Table 6.1. Shorter time to send a packet means that the device can stop the radio and go into power

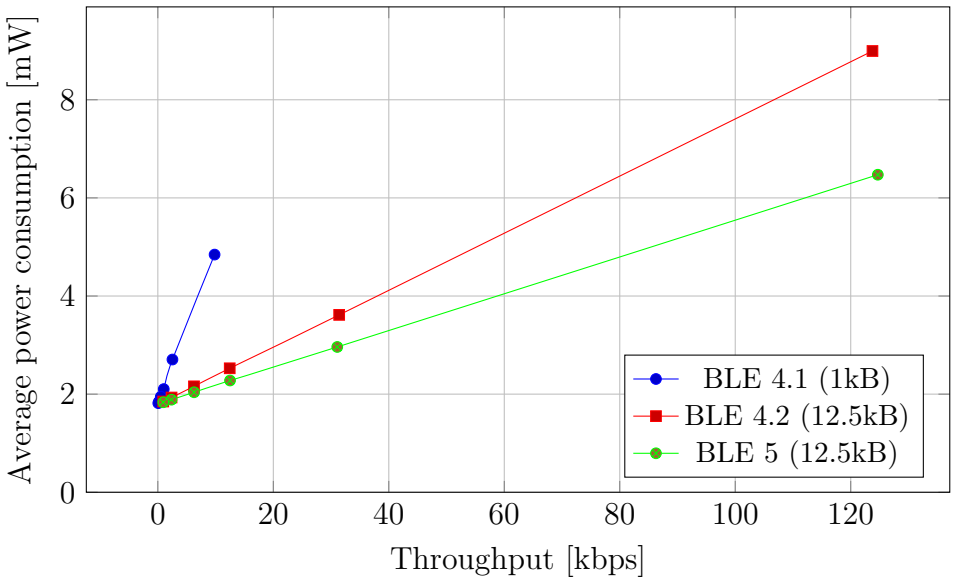

Figure 6.12: Mean average power consumption in relation to mean data throughput (obtained by changing the connection interval as described in Chapter 4.4) for write operation. Note: Data size and transmit time differ between BLE v4.1 and BLE 4.2 / BLE 5.

saving mode sooner, which has an impact on power consumption. Even with fragmented transmissions, as present in reading/writing operations this effect is stronger than the increase in momentary power consumption, thus improving overall power consumption.

Another interesting thing to observe is just how much BLE has evolved with v4.2 and 5, both in throughput and power consumption domains. This can be seen in Figure 6.13, where we can observe that the energy needed to transmit a certain amount of data lowers with each Bluetooth iteration. To make a fair comparison of BLE 4.1 with BLE 4.2 and BLE 5, the energy consumption for BLE 4.1 extrapolated from 1kB test energy consumption. The energy consumption measured with 1kB of transmitted data is multiplied by 12.5 to get the extrapolated values for 12.5 kB. The calculation for that is based on consumption invariance to data size, presented in Chapter 6.2.

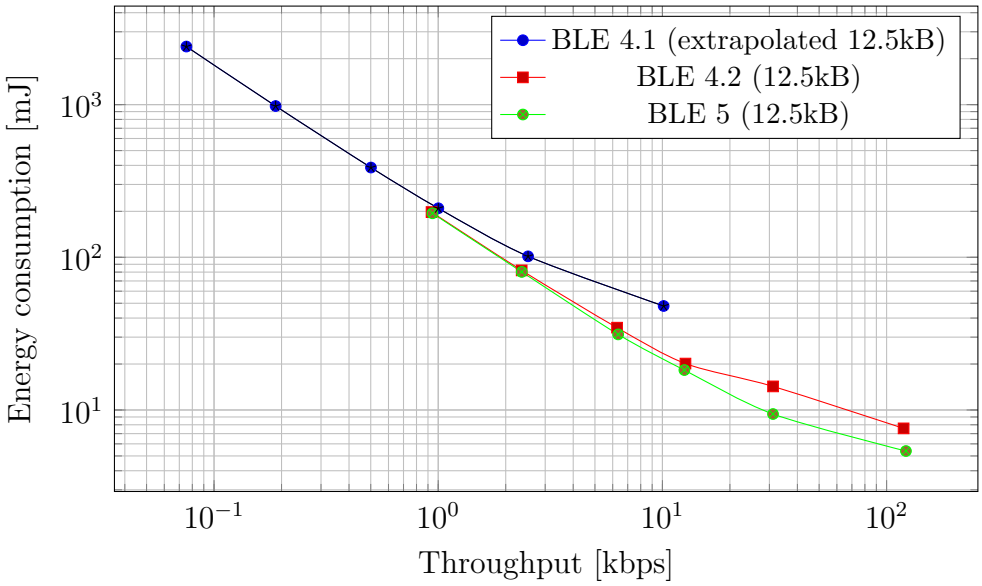

Figure 6.13: Mean energy consumption in relation to mean data throughput (obtained by changing the connection interval as described in Chapter 4.4) for read operation. Note: Energy consumption for BLE 4.1 extrapolated from 1kB test size.

6.5 BLE version comparison: notify and write without response

Notifications and writes without response are very similar operations, the only difference being the direction of data, which in our case does not play a significant role. They are the fastest operations that are supported by the majority of BLE devices. They are not bound by the limits of waiting for higher layer replies to commands. They still do need to respect the connection event start/end points, as well as the fact that if a packet fails to transmit in the current connection event, the connection event will be ended, and the failed packet will be retried in the next CE, with the queued packets being moved to the next CE as well.

With notify and write without response the differences between Bluetooth versions become much more apparent, as seen in Figures 6.14 and

6.15. Because with these two operations the connection events are filled with packets the differences in throughput, as well as consumption are much more apparent.

In Figure 6.14 we can observe the power consumption for different BLE versions. We see such clear separation between BLE v4.1 and v4.2 because of the increase of maximum *ATT_MTU* in BLE v4.2, which enables BLE v4.2 to have a much higher throughput. When comparing BLE v4.2 and BLE 5 we can see that they use different PHY layers, as the min/max power consumption values are no longer the same. Here we can see that BLE 5 2 Mb/s PHY consumes more momentary power, but it also nets a much higher throughput (roughly 1.7x increase).

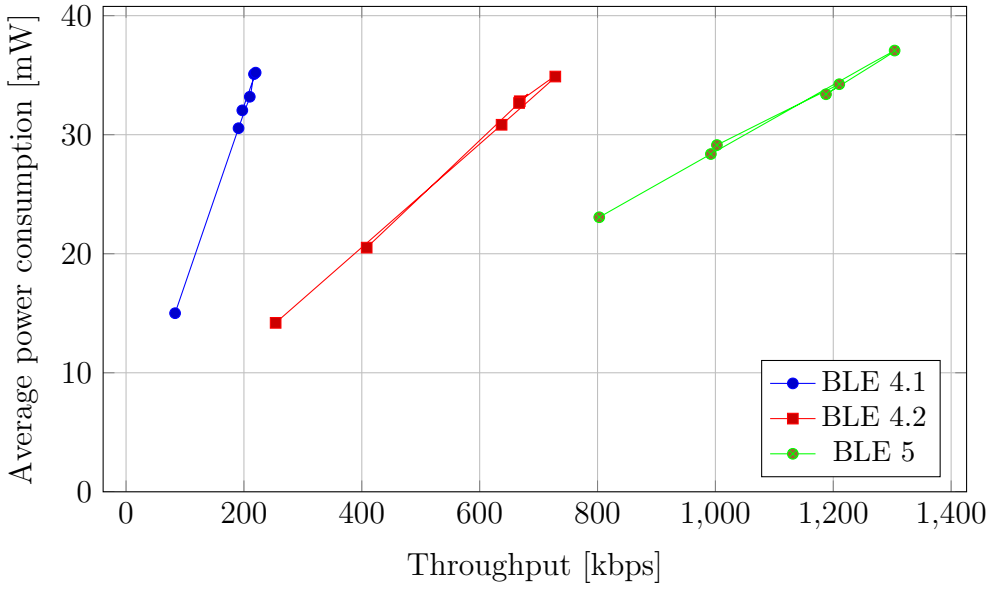

Figure 6.14: Comparison of BLE standards with notifications, shown as mean average power consumed for transmission in relation to mean throughput (obtained by changing the connection interval as described in Chapter 4.4)).

Once we look at Figure 6.15 it gets even more interesting, as we can see that the energy needed to transmit a single blob of data is lowered substantially with each BLE version. There is a significant difference in energy

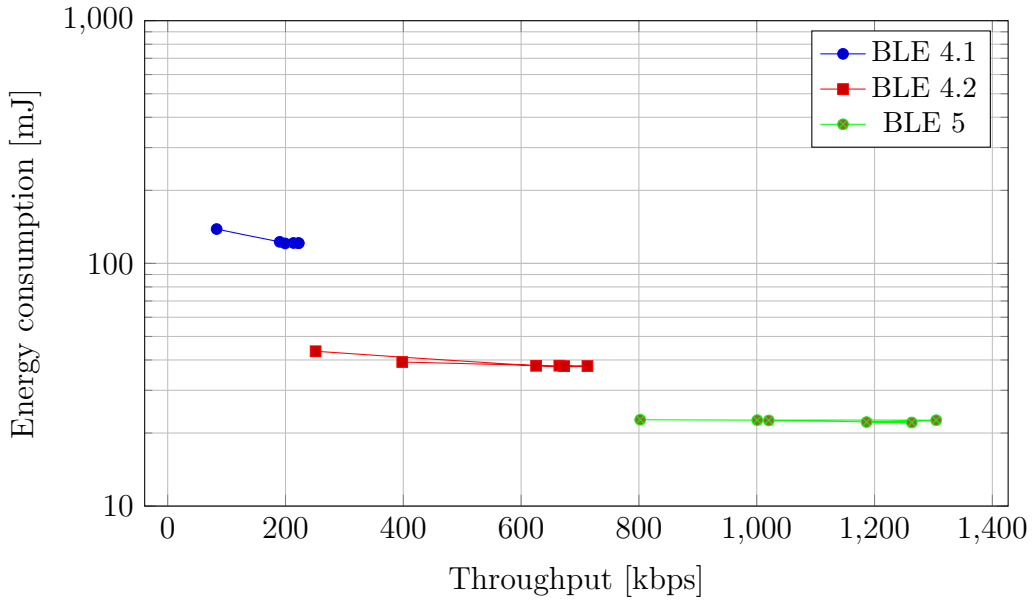

Figure 6.15: Comparison of BLE standards with write without response, shown as mean energy consumed for transmission in relation to mean throughput (obtained by changing the connection interval as described in Chapter 4.4)).

needed for fixed data transfer size between BLE 4.1 and BLE 4.2, which can be attributed solely to the increase of maximum *ATT_MTU* in BLE v4.2. This change decreased the number of packets to be sent for a set amount of data on the expense of each packet taking longer to send because they are now larger. That is a good thing because each packet needs some overhead, and an increase in packet size will decrease the ratio between overhead and packet size, which actually means fewer bits to transmit over the air. This nets a total decrease of power consumption, as well as the decrease of time to transmit the whole blob of data, which in turn decreases energy needed to do so because the radio is turned on for a shorter amount of time.

When we look at BLE 5, we can see it lowers the energy needed for transmission of the same amount of data even further. This can also be explained the same way as BLE v4.2 improvements, where we have again

lowered power consumption by means of shortening the amount of time the radio needs to be turned on, with the difference that for BLE 5 this is achieved by using the 2 Mb/s PHY. Even though we can see in Figure 6.14 that BLE 5 uses more power, in Figure 6.15 we can see that it uses less energy for transferring the same amount of data as BLE v4.2. This happens because the gains of shorter radio on time outweigh the increase of power consumption because of the use of a faster 2 Mb/s modulation PHY layer.

Another interesting and useful observation can be made when looking at energy consumption, and that is that BLE v4.2, and even more so BLE 5, appear to use the same amount of energy regardless of throughput (connection interval) selected. This information can be beneficial, as it enables us to tune our application to suit our needs in regards to throughput and latency of read/write commands without worrying about energy consumption.

Chapter 7

Conclusions

Recently introduced Bluetooth 5 promises twice the speed, four times the range and eight times increased broadcast capacity, all of which is supposed not to affect power consumption [8]. Because no articles were found that would compare the effect of different connection parameters and Bluetooth Low Energy versions on power consumption in relation to throughput, the comparison was therefore made in this thesis. We have delved into the Bluetooth core specifications to understand the inner workings of Bluetooth Low Energy devices, what the parameters that are available to the developer are and how do they affect throughput and consumption.

In this thesis, the work done to compare power efficiency of Bluetooth Low Energy versions is presented. This includes implementing and performing tests, as well as the measurements to get the data needed. Test cases were carefully selected to highlight the improvements offered by BLE versions. To be able to overcome the limitations of our power consumption measurement system we started by confirming that above a certain data size the average power consumption does not change, meaning we can compare the average power consumption of older BLE 4.1 with the newer and faster BLE 4.2 and BLE 5. This is important because of the limitation to capture at most 120 s of consumption data in one go. This posed a problem because the throughput of BLE versions is vastly different, so using large test data size

breached the limit with the older BLE 4.1 and using a small enough data size for BLE 4.1 when testing BLE 4.2 and BLE 5 proved to be too small to sufficiently stress the devices. Fortunately, because of the invariance of power consumption from data size, we could safely compare BLE standards even if we used different test sizes.

Next, we tested the effect of Connection Interval (see Chapter 3.3.1) on the throughput for the write and notify procedure, where we found that lowering the CI will increase throughput for writes (and reads), but that is not the case for notifications (and writes without response). For notifications, a CI that is too short will decrease the throughput because of extra overhead needed to open and close the connection event too often. On the other side, a connection interval that is too big will in practice decrease throughput because of interference and packet failures. The reason for decreased throughput, in that case, is that a connection event is closed when a packet failure happens and the packet is retried on next connection event, meaning that the rest of the closed connection event is unused, thus the average time of transmission increases with the increase of connection interval length.

Finally, we did the tests needed for comparing power efficiency in relation to throughput with different BLE versions and different operations. We grouped the comparisons by read/write operations and notify/write without response. That is because read and write are conceptually similar operations, with the difference being data flow direction, and the same goes for notify and write without response operations.

When looking at read/write power consumption, we can see that Bluetooth improves with each version, with the interesting thing being that BLE 5 further improves power consumption by using the 2 Mb/s PHY (see Chapter 3.2). That is interesting because using a higher speed PHY by itself consumes more power, but since it almost halves the time needed to transmit a packet it also means that it can turn the radio off sooner, hence decreasing power consumption. The reason for the decrease is that the power consumed by the 2 Mb/s PHY transciever is less than twice of the 1Mb/s PHY transciever.

That can also be seen when looking at total energy consumed to transmit the same amount of data (comparing BLE 4.2 and BLE 5), which decreases with BLE 5 because of the use of 2 Mb/s PHY.

The power consumption improvements of BLE 5 in relation to throughput become even more apparent once we look at results from testing notify and write without response. When looking at power consumption, we can clearly see that 2 Mb/s PHY of BLE 5 uses more power, with the benefit of almost doubling throughput. However, if we compare total energy needed to transmit the same amount of data we can clearly see that using 2 Mb/s PHY is the most efficient option when transmission of more substantial amounts of data is needed. Another interesting fact is that BLE 4.2 and BLE 5 actually consume practically the same amount of energy to transmit an amount of data, regardless of Connection Interval setting (and consecutively the throughput). This information can be beneficial, as it enables us to tune our application to suit our needs in regards to throughput and latency of read/write commands without worrying about energy consumption when large amounts of data need to be transmitted.

Decreasing the airtime of each packet is beneficial for lowering congestion rate of all the wireless technologies used in the environment. BLE 5 lowers the packets time on air when 2 Mb/s PHY is used, as the time needed to transmit each packet is halved. The new Channel Selection Algorithm #2 (CSA #2) is set to reduce the congestion rate in high interference environments even further by choosing the least used channels. Both of these effects should reduce congestion rate, which decreases power consumption even further (in high interference environments), as packet retransmissions become less necessary.

The findings in this thesis prove that there is no reason not to switch to developing and using devices that use Bluetooth Low Energy 5, as it brings higher throughput, extended range and better coexistence while using the same or lower amount of power for transmissions.

Bibliography

[1] M. Siekkinen, M. Hiienkari, J. K. Nurminen, and J. Nieminen, "How low energy is bluetooth low energy? comparative measurements with zigbee/802.15.4," in *2012 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, pp. 232–237, April 2012.

[2] K. Mikhaylov, N. Plevritakis, and J. Tervonen, "Performance Analysis and Comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI," *Journal of Sensor and Actuator Networks*, vol. 2, pp. 589–613, August 2013.

[3] S. Kamath, "Application Note AN092: Measuring Bluetooth Low Energy Power Consumption," 2010. Available: `http://www.ti.com/lit/an/swra347a/swra347a.pdf`, Acquired: 15.2.2019.

[4] M. Imran, "Power Measurements Techniques For Embedded Systems," 2015. Available: `http://apachepersonal.miun.se/~muhimr/index_files/Current%20measurements.pdf`, Acquired: 15.2.2019.

[5] Ž. Nakutis, "Embedded Systems Power Consumption Measurement Methods Overview," *MATAVIMAI*, vol. 44, no. 2, pp. 29–35, 2009.

[6] Nordic Semiconductor, "nRF52840 Development Kit," 2017. Available: `https://www.nordicsemi.com/eng/Products/nRF52840-DK`, Acquired: 27.11.2017.

[7] Nordic Semiconductor, "Power Profiler Kit," 2018. Available: https://www.nordicsemi.com/eng/Products/Power-Profiler-Kit, Acquired: 27.8.2018.

[8] M. Woolley, "Bluetooth 5 / Go Faster. Go Further," 2018. Available: https://www.bluetooth.com/bluetooth-technology/bluetooth5/bluetooth5-paper, Acquired: 1.3.2019.

[9] "Monsoon Solutions, Inc." Available: https://www.msoon.com/. acquired: 15.12.2018.

[10] F. Touati, O. Erdene-Ochir, W. Mehmood, A. Hassan, A. B. Mnaouer, B. Gaabab, M. F. A. Rasid, and L. Khriji, "An experimental performance evaluation and compatibility study of the bluetooth low energy based platform for ecg monitoring in wbans," *Int. J. Distrib. Sen. Netw.*, vol. 2015, pp. 1–12, Jan. 2016.

[11] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 2.0 + EDR*, 2004. Available: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=40560, Acquired: 5.8.2018.

[12] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 2.1 + EDR*, 2007. Available: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=456435, Acquired: 5.8.2018.

[13] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 3.0 + HS*, 2009. Available: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=456434, Acquired: 5.8.2018.

[14] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 4.0*, 2010. Available: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=456433, Acquired: 23.12.2017.

[15] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 4.1*, 2013. Available: https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=282159, Acquired: 23.12.2017.

[16] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 4.2*, 2014. Available: `https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=441541`, Acquired: 23.12.2017.

[17] Bluetooth SIG, *BLUETOOTH SPECIFICATION Version 5.0*, 2016. Available: `https://www.bluetooth.org/docman/handlers/DownloadDoc.ashx?doc_id=421043`, Acquired: 23.12.2017.

[18] CNX, "Bluetooth 5 Promises Four times the Range, Twice the Speed of Bluetooth 4.0 LE Transmissions," 2016. Available: `https://www.cnx-software.com/2016/06/10/bluetooth-5-promises-four-times-the-speed-twice-the-range-of-bluetooth-4-0-le-transmissions`, Acquired: 7.9.2018.

[19] R. Heydon, *Bluetooth Low Energy: The Developer's Handbook*. Prentice Hall, October 2012. ISBN: 978-0-13-288836-3.

[20] R. D. Akiba, C. Cufí, and K. Townsend, *Getting Started with Bluetooth Low Energy*. O'Reilly Media, Inc., May 2014. ISBN: 978-1491949511.

[21] Nordic Semiconductor, "nRF52840," 2017. Available: `https://www.nordicsemi.com/eng/Products/nRF52840`, Acquired: 27.11.2017.

[22] SEGGER Microcontroller, "J-Link-OB probe," 2017. Available: `https://www.segger.com/products/debug-probes/j-link/models/j-link-ob/`, Acquired: 27.11.2017.

[23] Nordic Semiconductor, "SoftDevices," 2017. Available: `https://www.nordicsemi.com/eng/Products/SoftDevices`, Acquired: 27.11.2017.

[24] SEGGER Microcontroller, "Real Time Transfer," 2017. Available: `https://www.segger.com/products/debug-probes/j-link/technology/about-real-time-transfer/`, Acquired: 27.11.2017.

[25] Gašper Kojek, "Master thesis measurement results." Available: https://github.com/ribafish/thesis_gasper_kojek_results, Acquired: 3.3.2019.

[26] git-scm.com, "Git Tools - Submodules," 2018. Available: https://git-scm.com/book/en/v2/Git-Tools-Submodules, Acquired: 7.9.2018.

[27] K. Ren, Bluetooth SIG, "Exploring Bluetooth 5 - How Fast Can It Be?," 2017. Available: http://blog.bluetooth.com/exploring-bluetooth-5-how-fast-can-it-be, Acquired: 7.9.2018.