Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Hálózati Rendszerek és Szolgáltatások Tanszék

Balogh András

NDN ALAPÚ SZOLGÁLTATÁS MEDIÁCIÓ MULTIHOP BLUETOOTH LOW ENERGY HÁLÓZATOKBAN

TDK Dolgozat

Konzulensek:

Dr. Imre Sándor, egyetemi tanár

Dr. Szabó Sándor, egyetemi adjunktus

2014

Tartalomjegyzék

Összefoglaló

A Bluetooth 4.1 szabvány megjelenésével (2013. december) az LE (Low Energy) variáns vezeték nélküli hálózatokban olyan új topológia-formációk kerültek bevezetésre az adatkapcsolati (MAC), melyek lehetővé teszik a több-ugrásos hálózatok felépítését. Bár ezen feladat megvalósításához léteztek korábban különböző megoldások, a technológia immáron a piconetek szétesése (vagy azok kikerülése) nélkül is lehetővé teszi a kommunikációt a kombinált (Master/Slave) szerepben való működés támogatásával.

Azonban a felsőbb rétegek (L2CAP, ATT, GATT) számára nem kerültek az említett szabvány által definiálásra a multi-hop útvonalak felderítésére, felépítésére, karbantartására és magára az adatforgalmazás mikéntjére vonatkozóan sem az interoperabilitáshoz szükséges különböző feltételek és szabályok. Ennek eredményeképpen bár léteznek jelenleg megoldások a több-ugrásos adatküldés megvalósítására, azok lényegüket tekintve nem veszik hasznát az L2CAP (Logical Link Control and Adaptation Protocol) feletti rétegeknek, így a definiált GATT (Generic Attribute Profile) keretrendszernek sem. Ebből fakadóan egy tőlünk két ugrásra levő modullal GATT specifikus üzenetek (pl. szolgáltatás-felderítés, adatok lekérdezése és írása, értesítések és jelzések beállítása) váltására csak igen körülményesen lehetünk képesek.

Az NDN (Named Data Networking, korábban Content-Centric Networking) koncepció és architektúra egy olyan új hálózati paradigmát követ, mely bizonyos értelemben szakít a hagyományos „párbeszéd” jellegű kommunikációs formával (így az állomáscímekkel és a hagyományos értelemben vett útvonalakkal) és a különböző tartalmak szétosztására helyezi hangsúlyt. Ennek megfelelően az adatok kerülnek címzésre (a nevük alapján) és nem az eszközök, így az adatok elérése a felsőbb rétegek számára sokkalta egyszerűbbé válik.

Dolgozatomban ezen megközelítés egyes elemeinek felhasználásával egy olyan szolgáltatás-közvetítési eljárást vázolok fel, mely az egyes modulokban található lokális GATT struktúrákat képes - a hálózat egészében - elosztott módon kezelni, különösebben összetett útvonal-választási eljárás alkalmazása nélkül, az egyes végpontok számára „kvázi” transzparens módon. Mindemellett a tesztelési és verifikációs célzattal létrehozott kísérleti rendszer is bemutatásra kerül, a megoldás valós környezetben való alkalmazásának értékelésével egyetemben.

Abstract

The latest version (v4.1) of Bluetooth technology (adopted in 2013. december) introduced the support of new topology formations to create multi-hop ad-hoc LE (Low Energy) links. Though this kind of operation could be achieved with various methods (i.e. Central and Peripheral fast-role-switching or using advertising events), these approaches broke the Link Layer „continuity" of operation, that is the „connected" piconets were not consistent. With the introduced combined-state Master/Slave operation in Bluetooth LE Link Layer, this kind of inconsistency can be avoided.

Nevertheless the modifications were done in the Link Layer, the upper layers (L2CAP, ATT, GATT) still lacks the functions, states and procedures (e.g. route-discovery, signaling, packet formats, etc.) needed for interoperability of these layers over multi-hop ad-hoc links. Because of this present methods for this kind of communication nowadays make little use of the layers over L2CAP (Logical Link Control and Adaptation Protocol), such as the defined GATT framework (e.g. service discovery, characteristic read and write, notifications, etc.). As a result GATT specific message exchange over more than one hop makes the existing protocols significantly more complex.

The NDN (Named Data Network, previously known as Content-Centric Networking) approach and architecture follows a new networking paradigm, in which the focus is moved from converstational type of communications (such as host addresses, and „links"), towards the content distribution. Because of this approach, the contents are addressed using their names instead, thus the data available in the network can be accessed in a more transparent way.

In this paper I propose a service mediation method, which was partially derived from the NDN concept, the gives the ability to manage the local GATT structures of the given devices in a distributed way, without complex routing schemes, quasi transparently throughout the network. In addition an experimental system is introduced, which was created for testing, verification and evaluation.

Bevezetés

A Bluetooth Low Energy (egyes terminológiákban Bluetooth Smart) technológia gyökerei egészen 2006-ig nyúlnak vissza, amikor is a Nokia Wibree néven bemutatott egy kifejezetten alacsony fogyasztásra tervezett kommunikációs technológiát [1]. 2010 júniusában a Bluetooth SIG (Special Interest Group) a technológia egy valamelyest módosított változatát integráltan elérhetővé tette a Bluetooth 4.0 szabványban, mely ettől fogva Bluetooth Low Energy megnevezés alatt él tovább [2].

A technológia térnyerése az első években (tapasztalatom szerint) visszafogott volt, noha az addig piacon levő Bluetooth modulok hardverein gyakorlatilag csak minimális mértékben kellett változtatni (azt is csak a fizikai rétegben, és csak akkor, ha nem támogatta a korábbi AMP kiterjesztést) a dual mode (hagyományos + LE) működés megvalósításához [2]. A technológia terjedésének a kulcsa az okostelefonok által nyújtott kompatibilitás volt. Az Apple Inc. készülékei (iPhone 4S kiadásától [3]) voltak az elsők 2011-ben, melyek ténylegesen támogatták a teljes Bluetooth LE (BLE) funkciókészletet egy nyílt API-n keresztül, ami újdonság volt, hiszen a hagyományos Bluetooth technológia kifinomult használatához az Mfi (Made-for-ixxx) programban való részvételre volt szükség, amelynek egyrészről pénzügyi vonzata volt, másrészről hardveres változtatást is igényelt az eredeti funkcionalitáshoz képest [4].

Az Android mobil operációs rendszer tekintetében már nem volt ilyen egyszerű az új technológia támogatásának integrációja. A 2013-as Google IO konferencián került hivatalosan beharangozásra [5], hogy az új Android verzió (4.3, API Level 18) már támogatni fogja a Bluetooth LE-t, ami az operációs rendszer megjelenésével valóban meg is valósult [6]. A csúszás feltételezhetően az új Bluetooth stackre való áttérésnek volt köszönhető, hiszen míg a 4.2-es verzió előtt a nyílt forráskódú BlueZ (4.93-as verzió) stackre hagyatkozott a rendszer (amelyben bár voltak BLE specifikus funkciók, azok működése saját tapasztalatomból fakadóan korántsem volt stabil) [7], addig ezen verziótól kezdődően a Broadcomm BlueDroid stackjére tértek át [8]. Így jelenleg a két legnagyobb piaci részesedéssel bíró mobil platform [7] immáron hivatalosan is támogatja a technológiát.

Tekintettel arra, hogy a Bluetooth LE technológia alapvetően kisebb adatok átvitelére tervezték (jellegéből fakadóan a szenzorhálózatok és a hagyományos adatátviteli hálózatok mezsgyéjén helyezkedik el), a burst-jellegű (pl. fájlátvitel) vagy a stream-jellegű (pl. hangátvitel) adatfolyamok csak szuboptimális módon vihetők át vele, így ezen megoldások terén nem történt előrelépés.

Azonban sorra jellennek meg olyan kisebb készülékek, amelyek a technológia adottságait (alacsony késleltetés, egyszerű robosztus eljárások, gyors kapcsolatkiépítés, alacsony fogyasztás) használják ki. Jelenleg számos gyártó kínál különböző kulcstartókat (amiket megtalálhatunk az okostelefonunk segítségével) [10], fitness készülékeket (melyek a sportolás közben monitorozzák a szívritmusunkat) [11], „okos" hőmérőket (amik értékeit szintén okostelefonnal olvashatjuk) [12], „okos" karórákat (amelyeken különböző értesítések jelenhetnek meg) [13] és még számos egyéb olyan eszközt, melyek ezen kisebb adatok befogadására és/vagy közlésére alkalmasak. Emellett a technológia kifejezetten alkalmas beltéri helymeghatározásra is, melyhez a különböző paramétereket (pl. adóteljesítmény, hirdetési periodicitás, stb.) szintén egy okos készülékről hangolhatjuk.

1. ábra – Bluetooth LE „gadgetek" [10]

Mindemellett az IoT (Internet-of-Things) hálózati paradigma [10] keretein belül az IETF (Internet Engineering Task Force) is kiemelt figyelmet fordít a technológiára, hiszen már 2011-ben megjelent az első draft, amely az IPv6-alapú kommunikációt valósítja meg Bluetooth LE linkek felett, és amelynek kidolgozása jelenleg is folyamatban van [11].

A Bluetooth 4.1-es verziószámú szabvány megjelenésével a technológiában új topológiák kerültek bevezetésre, melyeket a szabvány a hagyományos technológiából is ismert „scatternet" (több összekapcsolt piconet) névvel illet [12]. Azonban hasonlóan a hagyományos változathoz, a GATT funkciók multihop interoperabilitásához szükséges alsóbb rétegbeli funkciók nem kerültek bevezetésre, minek következtében a korábban felsorolt eszközök együttes kezelésének transzparens mivoltához az eszközök hatótávolságán belül kell lennünk (ami tapasztalatom szerint beltérben

hozzávetőlegesen ~20m, NLOS terjedés esetén, irodai környezetben), ami összességében erősen korlátozza, valamint adott esetben el is lehetetleníti a kiterjedtebb rendszerek megvalósítását, így egy olyan megoldás kidolgozása, mely lehetővé teszi a többes ugrásos kommunikációt, mindenképpen indokolt.

Noha a szabvány 2013 decemberében már napvilágot látott [16], vajmi kevés publikáció született eddig ezen képesség modellezésével és megvalósíthatóságával kapcsolatban. Így megoldásom kidolgozása során ezekre nem is hagyatkozhattam.

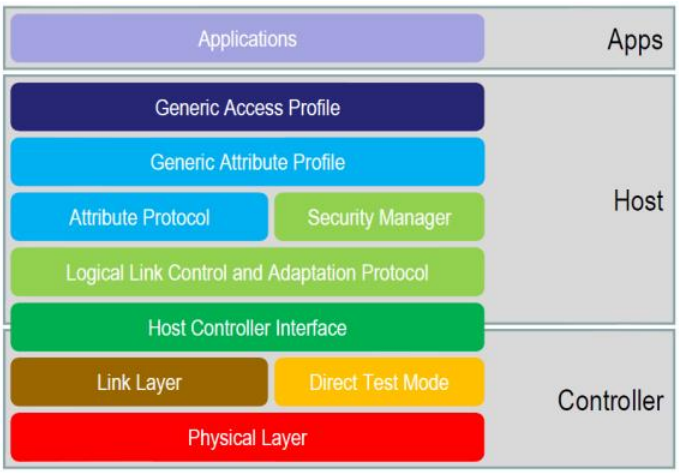
Dolgozatom során összefoglaló jelleggel bemutatásra kerül a technológia, valamint egyetlen publikált megoldás, amely a Bluetooth LE alapú multihop kommunikáció megvalósítására eddig született (I. pont). Ezt követően kitérek az NDN (Named Data Networking) megközelítés és architektúra azon lényegesebb elemeire, amelyek a megvalósítás alapötletét adták (II. pont). Az ismertetést követően bemutatom az általam tervezett szolgáltatás mediációs szolgáltatást, mely ponton belül kitérek annak előnyeire és az alkalmazhatóság vizsgálatára is (III. pont). Ezután összefoglalom a kísérleti implementáció fontosabb mozzanatait, majd verifikálom a megoldást a kísérleti rendszer PoC (Proof-of-Concept) jellegű működésének értékelésével (IV. pont). Mindezek után kitérek a megoldás működéséből és jellegéből fakadó olyan új lehetőségekre, melyek alkalmazása megfontolandó lehet mind a Bluetooth Low Energy, mind egyáltalán a hálózatok világában (V. pont). Dolgozatomat végül konklúziómmal zárom.

I. A Bluetooth Low Energy technológia áttekintése

Az alábbi alpontokban összefoglaló jelleggel bemutatásra kerül a Bluetooth technológia Low Energy (LE) kiterjesztésének architektúrája, valamint azt követően kitérek néhány olyan (eddig publikált, vagy triviálisan következő) lehetőségre, amelynek kihasználásával megvalósítható lehet a többes ugrásos kommunikáció ezen technológiában.

1. Architektúra

A Bluetooth LE stack rétegei alapvetően három csoportra tagozódnak (Application, Host és Controller), ugyanakkor ez csak egy lehetséges, inkább informatív leképezés, mintsem követelmény, hiszen számos SoC (System-on-Chip) megoldás van jelenleg a piacon (Texas Instruments, Cambridge Silicon Radio, Nordic Semiconductors, stb.), melyek egyazon integrált áramköri lapkán valósítják meg az összes réteget [17].



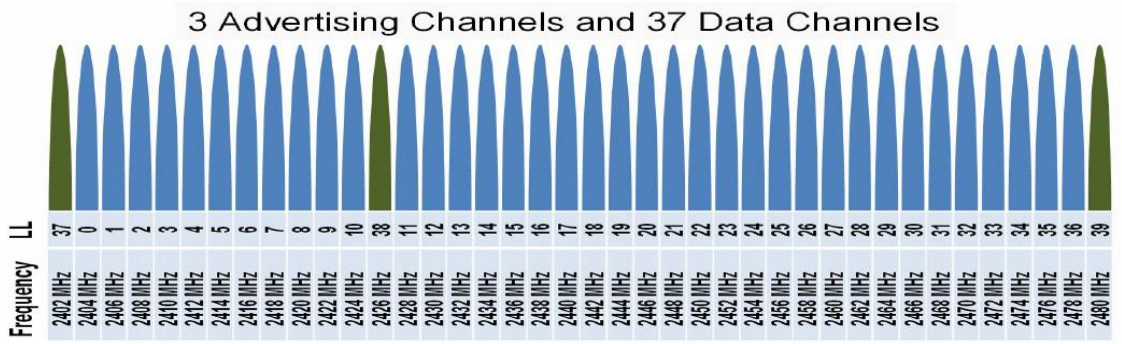
2. ábra – Bluetooth LE architectúra [18]

Az alábbi pontokban ezen rétegek működése kerül áttekintő jelleggel bemutatásra.

1.1. Fizikai réteg (PHY)

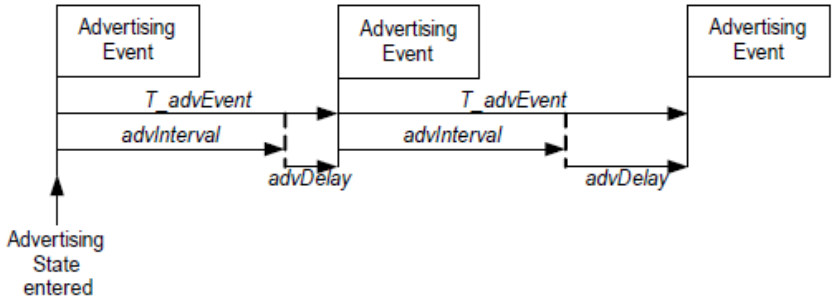
Egy Bluetooth LE eszköz a 2,4 GHz-es ISM sávban működik. Mivel a sávban más technológiák is megtalálhatók (WiFi, ZigBee), az esetleges interferenciákat és átlapolódásokat egy adaptív frekvenciaugratási mechanizmussal (A-FHSS - Adaptive Frequency Hopping Spread Spectrum) kerüli el. Bináris frekvenciamodulációt alkalmaz, típus szerint a GFSK-t (Gaussian Frequency Shift Keying), ami gyakorlatilag

megegyezik az egyszerű FSK-val. Annyi a különbség, hogy az alapsávi bináris impulzusokat egy Gauss-szűrőn vezetik át, ami a nagyfrekvenciás komponenseket kiszűri és így simább (ugrásoktól mentes) jelalak kerülhet modulálásra [2].



3. ábra – Frekvenciasávok a Bluetooth LE technológiában [18]

A technológia több hozzáférési sémát alkalmaz, nevezetesen a frekvenciaosztásos (FDMA) és az időosztásos (TDMA) sémákat. 40 db fizikai csatornát határoz meg a frekvenciatartományban a specifikáció: 3 db hirdető és 37 db adatcsatornát, melyek vivői egymástól 2 MHz-re helyezkednek el. A TDMA rendszer határozza meg azt, hogy melyik eszköz melyik időpontban küldhet csomagokat. A fizikai csatornát ezen időegységek osztják részekre, melyeket a specifikáció „event”-eknek nevez. (A következő oldalakon az esetlegesen nem egyértelmű fordítások elkerülése végett a specifikációban használt angol szavakat fogom alkalmazni.) Kétféle „event”-et különböztetünk meg: „Advertising event” és „Connection event”. Az *Advertising event*-eket tipikusan hirdetésekre alkalmazzuk, míg a *Connection event*-eket a megbízható adatküldésre [2].



4. ábra – Advertising Event [17]

Példaképpen, több egymást követő „Advertising event” esetén az időbeni megkötések a következőképpen alakulnak [2]:

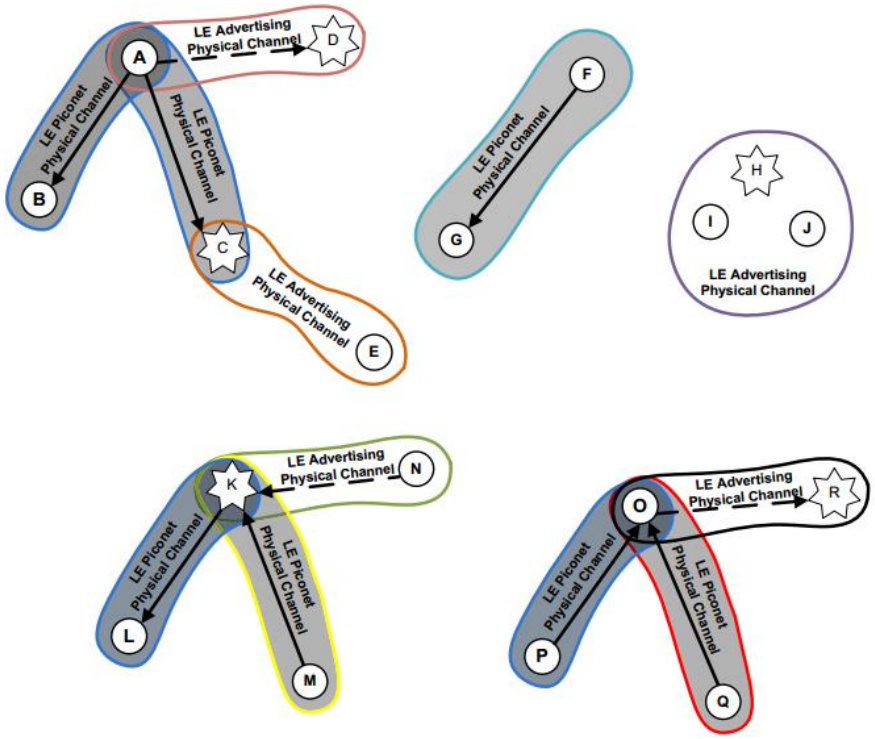
$20ms < advInterval < 10{,}24s$ (tetszőlegesen állítható)

$0ms < advDelay < 10ms$ (pszeudorandom érték)

1.2. Link Layer (MAC)

A fizikai csatorna (PHY) felett a réteghierarchiában a fizikai kapcsolatok (MAC) és további felsőbb szintű csatornák, valamint az ezekhez tartozó vezérlő eljárások találhatók. A hagyományos technológiához hasonlóan itt is egy mester (master) egységhez több szolga (slave) egység tartozik, ugyanakkor alapvető eltérés, hogy immáron nem egyetlen frekvencia-ugratási szekvencia szerint hangolják a (piconetben részt vevő) slave egységek a masterrel szinkronban a frekvenciájukat, hanem minden egyes slave-master kapcsolathoz külön fizikai csatorna, s így ugrási szekvencia tartozik, melyet ugyanakkor továbbra is a mester egység határoz meg (mindemellett egy slave, egység kérheti a szekvencia módosítását) [17].

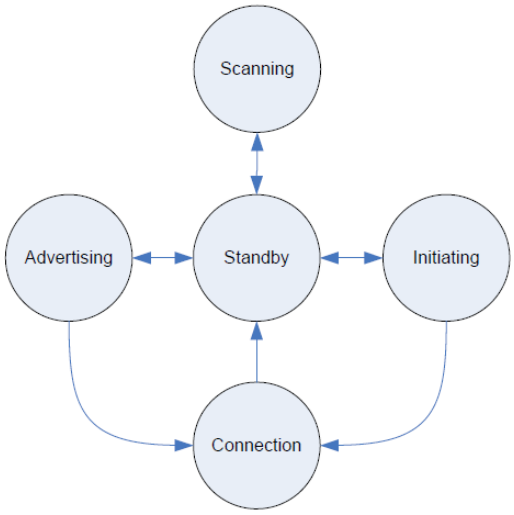
Mindez összességében azt jelenti, hogy a maximum 8 élő (7 slave + 1 master) eszközt megengedő TDM struktúra immár a múlté, hiszen a mester egység maga osztja be az eszközöket a rendelkezésre álló erőforrások függvényében (akár 255-öt).



5. ábra – Lehetséges topológiák a Bluetooth LE hálózatokban [17]

A fizikai csatornát egy vagy több aszinkron adatátvitelt támogató logikai csatorna adatának átvitelére alkalmaz a rendszer. A fizikai csatornák és kapcsolatok kezelésére a Link Layer Protocol (LL) hívatott, melynek paraméterei és vezérlései a

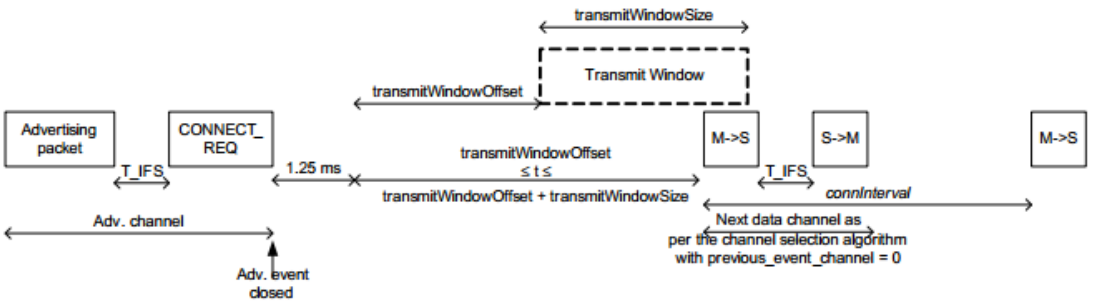
felhasználói adatokkal közös csatornán kerül átküldésre. Azon eszközök, amelyek aktív szerepet töltenek be egy piconet-ben alapértelmezés szerint egy LE Asynchronous Connection Logical transport (LE ACL) entitással bírnak. Az ezen állapotokhoz tartozó jelzések átvitele egy ilyen LE ACL-en történik [17].



6. ábra – Link Layer állapotgép [17]

A Link Layer működése egy állapotgéppel írható le. Az egyes állapotokban a fizikai kapcsolati réteg (Link Layer) a következő feladatokat látja el:

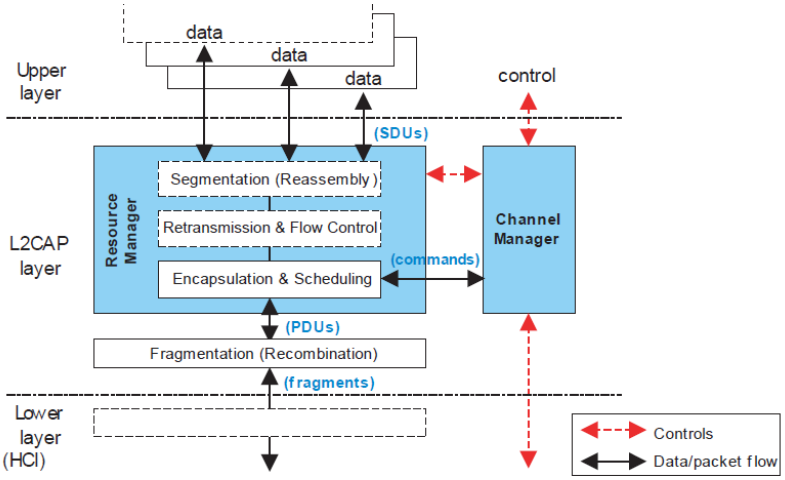
- **Standby** állapotban a Link Layer sem nem küld, sem nem fogad csomagokat.
- **Advertising** állapotban a Link Layer a hirdetési csatornákon küld csomagokat, illetve egyes esetekben ezeken a csatornákon hallgatózik és válaszol is bizonyos csomagokra. Advertising állapotban levő eszközt Advertiser-nek is nevezi a későbbiekben a specifikáció.
- **Scanning** állapotban a Link Layer a hirdetési csatornákra hangolódva hallgatózik az Advertiser-ek által küldött csomagok után. A Scanning állapotban levő eszközök a Scanner-ek.
- **Initiating** állapotban a Link Layer kitüntetett eszközök csomagjai után hallgatózik a hirdetési csatornán, és adott esetben ezekre a hirdetésekre válaszolva kezdeményezi a kapcsolatok létrehozását. Iniator-oknak is hívja a későbbiekben a specifikáció az ebben az állapotban lévő eszközöket.
- **Connection** állapotban egy eszköz kétféle szerepet tölthet be: master vagy slave. Akkor kerül a Link Layer ebbe az állapotba, ha van kiépült kapcsolata egy másik eszközzel.



7. ábra – A kapcsolatfelépítés folyamata [2]

A Link Layerben mennek végbe a címzéssel kapcsolatos mechanizmusok is. Változás a hagyományos technológiához képest, hogy a címzést immáron sokkal szabadabban végezhetjük. A Bluetooth LE-ben ugyanis bevezetésre kerültek a random és privát címek, míg ez előbbi egyszerű pszeudo-véletlen algoritmusokkal szabadon generálható, addig az utóbbi biztonsági célokat szolgál. Nevezetesen, ha párosítottuk magunkat egy eszközhöz, úgy annak címét teljesen elrejthetjük mások elől, hiszen a stack véletlenszerűen fogja adott időközönként változtatni azt, és a csak a közös titok birtokában lehetünk képesek olyan címet előállítani a kapcsolódás során, melyre az adott modul válaszolni fog [2].

1.3. Az L2CAP réteg



8. ábra – Az L2CAP réteg főbb feladatai [17]

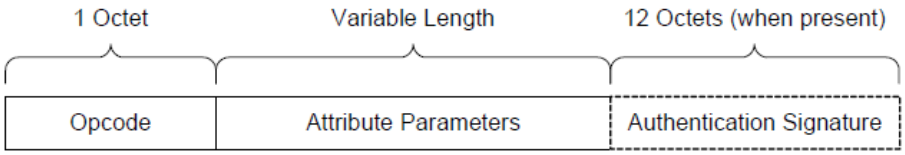
A Link Layer fölött helyezkedik el az L2CAP (Logical Link Control Adaptation Protocol), amely csatorna alapú absztrakciót lát el a szolgáltatások és alkalmazások számára. Ez az eljárás végzi el a szegmensek darabolását és a darabok összeállítását, illetve a logikai csatornák multiplexálását és az adatfolyam vezérlését a megosztott LE ACL-en. A protokollnak van egy kitüntetett LE ACL-je, amin csak a protokoll-

specifikus üzenetek kerülnek elküldésre. A protokoll gyakorlatilag megegyezik a korábbi Bluetooth eszközök által használttal, azonban annak csak egy részét használja az LE rendszer [2].

1.4. Az ATT és SMP protokollok

Az L2CAP fölött helyezkednek el, egymás mellett az Attribute Protocol (ATT) és a Security Manager Protocol (SMP). Amíg az utóbbi fix L2CAP csatornát használ a biztonsági szolgáltatások nyújtására (párosítás, kulcskezelés, hash generálás stb.), addig az Attribute Protocol feladata a kisebb méretű adatokkal való kommunikáció megvalósítása, illetőleg a más eszközök képességeinek és szolgáltatásainak felderítése, amelyek szintén egyetlen kötött L2CAP csatornán történnek [2].

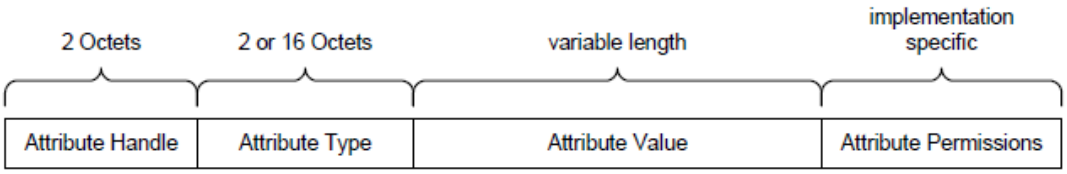
Az ATT a következő PDU (Protocol Data Unit) formátumot határozza meg:



9. ábra – Attribute Protokoll PDU [17]

Az *opcode* mező határozza meg, hogy milyen jellegű PDU-ról beszélünk. Értéke jelölhet kérést, választ, jelzést, értesítést vagy megerősítést. Az *Attribute Parameters* tartalmazza az *opcode* által kijelölt művelethez tartozó adatot (*Attribute*). Az *Athentication Signature* mező pedig opcionálisan használható autentikációs célokra.

Egy *Attribute*-ot négy elem határoz meg:

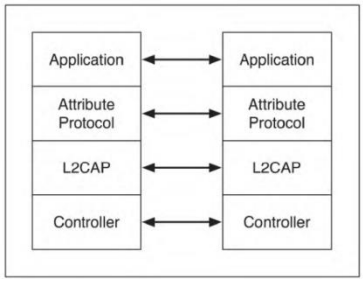


10. ábra – Attribute-ok attribútimai [17]

Az *Attribute Handle* egyfajta indexként szolgál, amely kijelöli az adott *Attribute*-ot a belső adatbázisban. Az *Attribute Type* egy UUID (Universally Unique Identifier), amelyet az *Attribute* megjelölésére használ a rendszer. Az *Attribute Value* az az adat, amelyet a *Handle* jelöl ki és a *Type* jelöl meg. Az *Attribute Permissions* határozza meg az egyes műveletekhez szükséges jogosultságokat a kitüntetett *Attribute*-ra nézve [2].

Az *Attribute*-ok lényegére, illetve jelentésére a következő réteg, a GATT réteg kifejtése során derül fény.

1.5. A GATT (Generic Attribute Profile) keretrendszer



11. ábra – A GATT alatti rétegek [17]

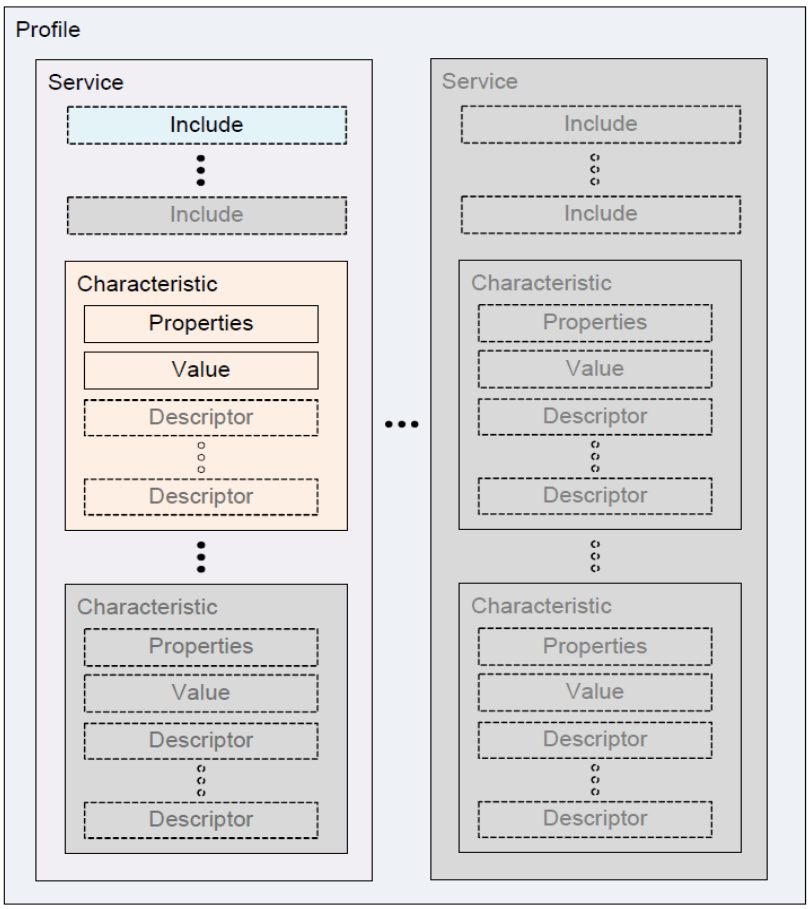
A Generic Attribute Profile (GATT) egy szolgáltatás keretrendszert definiál, amely az Attribute Protocol-t alkalmazva lehetővé teszi az egyes szolgáltatások (*Services*) felderítését, illetve az ezekhez a szolgáltatásokhoz tartozó adatok/tulajdonságok (*Characteristics*) olvasását és írását egy távoli eszközön (*Peer*). Alapvetően arra találták ki, hogy egy olyan alkalmazást vagy egy további profilt építhessünk rá, melyben kliens és szerver közti kommunikációt valósítunk meg [17].

A GATT-ot implementáló eszközök számára a következő szerepeket definiálja a profil:

- **Kliens:** Ez az eszköz kezdeményezi az egyes parancsokat és kéréseket, és képes fogadni a válaszokat, az értesítések és jelzések mellett a szervertől
- **Szerver:** A szerver fogadja a kéréseket és parancsokat, valamint válaszol rájuk, továbbá értesítéseket és jelzéseket is küld a kliens számára

Ezek a szerepek azonban nincsenek eszközhöz kötve. Csak egy adott feladat során töltik be ezt a szerepet, a feladat elvégzésével azonban kilépnek belőle. Egy eszköz kétféle szerepet is betölthet egyidejűleg [2].

A GATT meghatároz egy olyan struktúrát, amelyben a profil-specifikus adatokat tárolja. Minden alapelem egy attribútumot jellemez, tehát az attribútumok, amelyeket az Attribute Protocol alkalmaz, tartalmazzák ezeket az adatokat. Az egyes alapelemeket a nekik megfelelő definícióval kell leírni, amely a specifikációban részletesen dokumentált. Az alkalmazott struktúrát a következő ábra szemlélteti [2].



12. ábra – A GATT által definiált struktúra [2]

1.5.1. Service

Egy adatállomány és az azokhoz tartozó működési módok, amelyek egy adott funkciót valósítanak meg. *Service*-okat *Service definition*-nel definiálunk, amely tartalmazhat egyéb kapcsolódó *service*-okat, kötelező *Characteristics*-eket és opcionális *Characteristics*-eket. Összesen kétféle *service*-ot különböztetünk meg: *primary* és *secondary*. Egy *primary* service jellemzi az adott profil (alkalmazás) alapvető szolgáltatásait, szerepelhet kapcsolódó *service*-ként is egy másikban, de önmagában is működőképes. Ellenben a *secondary* service-ok csak akkor lehetnek használatban, ha valamilyen kontextusban szerepelnek (tehát egy másik *service*-hoz kapcsolódnak), egyéb esetben inaktívak [2].

1.5.2. Included Service:

Ezen mezők segítségével hivatkozhatunk más *service*-okra és kapcsolhatjuk azokat a jelenlegihez [2].

1.5.3. Characteristic:

Ez az elem szolgáltatja azt az aktuális értéket, amely az adott szolgáltatásban a felsőbb rétegek számára (alkalmazás) nyújt információt. Azonban egy *Characteristic*-et jellemez a formátuma és a tulajdonságai is, így a definíciónak a következő deklarációkat kell magában foglalnia: *Characteristic Declaration*, *Characteristic Value Declaration* [2].

1.5.4. Descriptor:

A deszkriptorok a hierarchia legalsó szintjén foglalnak helyet, segítségükkel olyan kiegészítő információkkal és funkciókkal láthatunk el *Characteristic* értékeket, melyekkel azok hozzáférhetősége, „viselkedése" (jelzések, értesítések), értelmezése és felismerése könnyebbé tehető [2].

A GATT által nyújtott legfontosabb szolgáltatások:

Attribute Caching:

Az Attribute caching egy olyan optimalizációs módszer, amely lehetővé teszi az egyes attribútumok megjegyzését, így újrakapcsolódáskor nem kell újra lekérdezni az összes szükséges struktúrát. A specifikáció értelmében, csak a Handle és a hozzá tartozó Characteristics értékeket kell cachelni. Az attribútumokhoz tartozó handle-ök a szerverben nem változhatnak (jelzés nélkül) az idő során [2].

Amikor valami megváltozik a szerver által támogatott service-okban, a szerver jelzést küld az összes kliensnek, amennyiben éppen nem tudja őket elérni, akkor megvárja, amíg kapcsolódnak hozzá és akkor küld jelzést, ha megbízható kapcsolattal rendelkeznek. Amennyiben nem megbízható, akkor újrakapcsolódáskor egy service discovery-t kell végrehajtaniuk, amelynek folyamatára a későbbiekben ki is térek [2].

Jelzéskor a szerver egy Handle Value Indication-t küld a klienseknek, melyben kijelöli azokat a Handle-ket, amelyek már nem aktuálisak vagy változott a tartalmuk. A kliensek ekkor megjelölik ezeket a handle-öket az attribute cache-ükben, és egészen amíg service discovery-t nem hajtottak végre, a felsőbb rétegek nem férhetnek hozzá ezekhez az adatokhoz. A szerver a Handle Value Confirmation-t üzenet fogadásával tudja meg, hogy az adott kliens már tud a változásokról [2].

Miután a kapcsolat kiépült és az autentikáció (ha szükséges) megtörtént, az egyes értékek egy egyszerű adatbázis lekérdezés jelleggel olvashatók (vagy írhatók) [2].

Primary Service Discovery:

Ezt a procedúrát akkor alkalmazzuk, amikor a szerveren található primary service-okat szeretnénk felderíteni. Miután ezeket felderítettük egyéb procedúrákat alkalmazunk a service-okkal kapcsolatos egyéb információk felderítésére. Ha az összes primary service-t fel akarjuk deríteni, úgy ezen procedúra használatos [2].

Characteristic Discovery:

Az eljárás során kideríthetjük egy adott service-on belül található Characteristic típusú adatokat. Lehetséges mind UUID-ra való egyezés keresésére, mind egy adott handle tartományban való összes ilyen deklaráció felderítésére [2].

Characteristic Value Read:

Egy Caharacteristic érték olvasása többféleképpen megtörténhet. Amennyiben csak az UUID-t tudjuk, úgy az alapján kerül az első egyezés értékének kiolvasásra, amennyiben viszont tudjuk a pontos helyét a GATT adatbázisban, úgy a megfelelő handle alapján kiolvasható. Megjegyzendő, hogy mindkét esetben csak maximum ATT_MTU-1 méretű értékeket olvashatunk, azonban lehetséges a hosszabb értékek olvasása is egy Blob jellegű struktúra használatával, ekkor azonban tudnunk kell azt a Handle tartományt, ami lefedi az adott értéket. Lehetséges továbbá több Characteristic Value egyúttal való olvasására is [2].

Characteristic Value Write:

Ez az eljárás valósítja meg az egyes Characteristic Value-k írását a szerveren. Ahogyan az olvasásnál, itt is többféleképpen járhatunk el. Írhatunk visszajelzés nélkül (egyszerű írás esetén visszajelzést kapunk a szerver felől), lehetőségünk van aláírt írásra is, melyre alapvetően akkor van szükség, ha az ATT protokollt hordozó réteg nincs titkosítva, de megbízható kapcsolattal rendelkezik a kliens a szerver irányában. Ezen felül hosszú Char. Value-kat is lehetőségünk van írni, az előbbiekben ismertetett olvasás funkcióval analóg módon. Mindezen felül megbízható írást is alkalmazhatunk, melynek lényege, hogy az adatokat az ATT protokoll a válaszokban felülvizsgálja. Továbbá alkalmazható a funkció több Characteristic egyúttal való írására is (szintén az olvasással analóg módon) [2].

Characteristic Value Notification:

Ez a funkció lehetőséget ad arra, hogy értesíthessük a klienseket egyes Characteristic Value-k értékéről és nem várunk visszajelzést arról, hogy a kliens ezt tudomásul vette-e. Ami felfogható egy best-effort jellegű adatküldésként is [2].

Characteristic Value Indication:

Indication esetén ugyanaz történik, mint Notification esetén, azonban visszajelzést is várunk a kliens felől. Ami az előző analógia mentén megbízható adatküldésre ad lehetőséget [2].

1.6. A GAP (Generic Access Profile) réteg

A Generic Access Profile definiálja azokat az általános procedúrákat, amelyek alkalmazásával az egyes Bluetooth LE eszközök felderíthetők, továbbá szabályozza a kapcsolatok kezelésének aspektusait, illetőleg az egyes biztonsági szintek alkalmazásának módszereit. Az egyes eszközök együttműködéséhez a profilban kötelezővé tett (*mandated*) lépések és azok formai megkötései minden eszközre egyaránt vonatkoznak. Az opcionálisnak megjelölt procedúrák azonban természetesen nem [2].

Egy eszköz a következő GAP szerepeket töltheti be (párhuzamosan akár többet is) [2]:

- Broadcaster
- Observer
- Peripheral
- Central

1.6.1. Broadcaster

Egy, a Broadcaster szerepben működő eszköz Advertising event-eket küld, a következő Link Layer-beli előírások szerint [2]:

Link Layer-beli állapot: **Advertising**

1.6.2. Observer

Observer szerepben egy eszköz az Advertising event-ek után hallgatózik, a hirdetési fizikai csatornákon a következő Link Layer-beli előírások szerint [2]:

Link Layer állapot: **Scanner**

A Scanning során a Scanner a hirdetési csatornákon hallgatózik a scanWindow által kijelölt időintervallumban. A scanInterval pedig azt mondja meg, hogy két scanWindow kezdete között mennyi idő teljen el. A scanWindow értéke kisebb, vagy egyenlő 10,24 másodperccel. A scanInterval pedig értelemszerűen legalább annyi, mint a scanWindow értéke. Miután a scanWindow által kijelölt idő letelt, a következő ablak a soron következő hirdetési csatornán kezdődik [2].

1.6.3. Peripheral

Bármely olyan eszköz, amely elfogadja egy LE fizikai kapcsolat létrehozását, Peripheral szerepet tölt be. Egy ilyen eszköz a Link Layer **Connection** állapotában **Slave** szerepet fog betölteni. Ez a szerep több állapotot is felölel a Link Layerben, ugyanis **Advertising** állapotból indul, és egy CONNECT_REQ hatására a **Connection** állapotbeli **Slave** szerepbe megy át [2].

1.6.4. Central

Olyan eszközök, amelyek támogatják ezt a szerepet, kezdeményezhetik egy LE fizikai kapcsolat létrejöttét. Az ilyen eszközök a Link Layer **Connection** állapotában **Master** szerepet fognak betölteni [2].

Az előzőhöz hasonlóan ez a szerep is több állapotot érint a Link Layer-ben. **Initiating** állapotból indul, majd egy Connectable Advertising event hatására CONNECT_REQ PDU-t küld az **Advertiser** állapotban levő **Peripheral** eszköznek, ezt követve az **Initiating** állapotból a **Connection** állapotba lép [2].

A Central szerepben levő eszközök egyben a felderítési procedúrákat is támogatják. Ellentétben a korábbi Bluetooth verziókkal, ahol egy ilyen folyamat akár 4-5 másodpercet (több eszköz esetén még többet) is igénybevett, a felderítés és kapcsolódás itt igen egyszerű (elméletben akár 200-300ms is elég lehet) [2].

1.7. Alkalmazási réteg

A réteghierarchia legfelsőbb szintjén maga az alkalmazás található, amely meghatározza azokat a funkciókat (features), amelyeket a Bluetooth technológia segítségével végzünk el. Ebbe a rétegbe tartoznak azok a Bluetooth SIG által előredefiniált profilok is, amelyeket sok esetben a technológiát elérhetővé tevő SoC-ok már implementáltan tartalmaznak (pl. Proximity Profile) [2].

2. Megoldások a többes ugrásos BLE kommunikációra

Az alábbi pontokban olyan vélt és valós megoldásokat ismertetek összefoglaló jelleggel, melyek a Bluetooth Low Energy scatternet formációinak kiépítését, és az azon való kommunikációt teszik lehetővé. A technológia jellegéből fakadóan csak két megközelítést alkalmazhatunk erre a célra. Az egyik az, hogy az alkalmazási rétegben definiálunk különböző metódusokat, melyek az alsóbb szintek funkcióinak végrehajtásával megvalósítják a kívánt működést. A másik megközelítés, hogy az alsóbb rétegek működését módosítjuk a scatternetek-en kialakítható linkek absztrakciójával úgy, hogy a felsőbb rétegek számára ezen kapcsolatok transzparens módon működjenek.

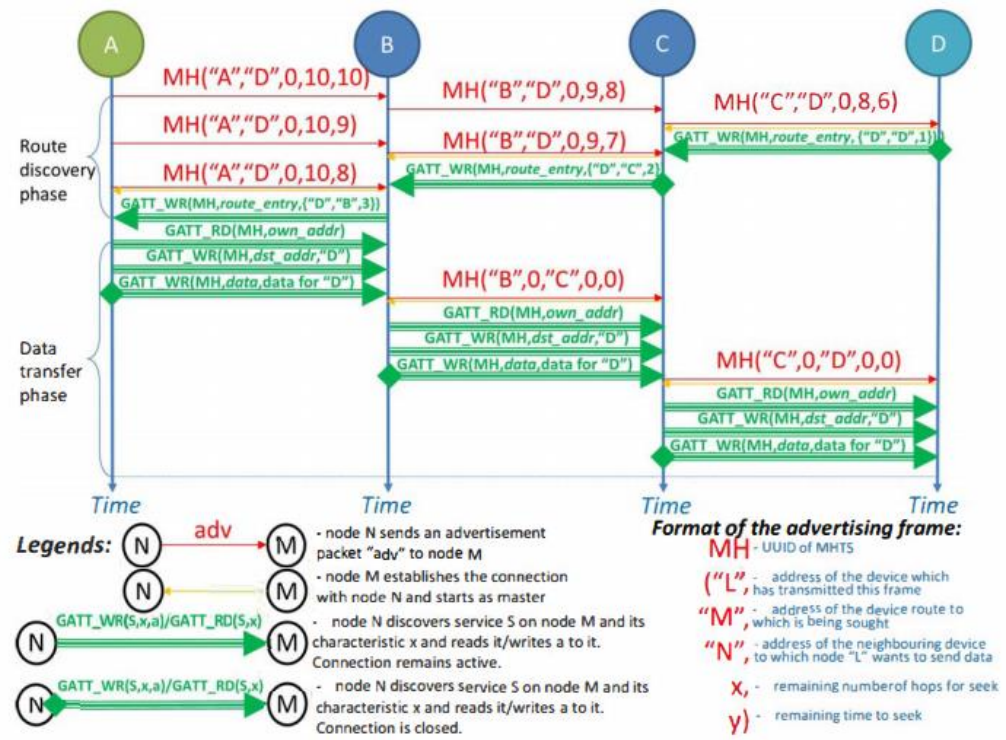
2.1. Multihop kommunikációs GATT szolgáltatás

2013 novemberében a Finnországban található Oulu-i egyetem kutatói egy olyan megoldással álltak elő, mely egy definiált GATT szolgáltatás segítségével valósította meg a multihop kommunikációt [19].

Characteristic	Permissions	Size, bytes	Description
route_entry	Write	14	Used to transfer data about the routes. Includes address of the target node, address of the next-hop node and the total number of hops to the target node.
own_addr	Read	6	Address of this node.
dst_addr	Write&Read	6	Address of the final target device for the block of data.
data	Write	>20	Part of data to be sent to *dst_addr*.

13. ábra – A definiált multihop kommunikációs GATT szolgáltatás szerkezete [19]

Az ad-hoc multihop kommunikációnak általánosságban véve alapvetően két fontosabb részét különböztethetjük meg. Az első a lehetséges útvonal(ak) felderítése a céleszköz irányába, a másik az útvonal kiválasztása, és az azon való kommunikáció biztosítása.



14. ábra – Az útvonal felderítése és a kommunikáció folyamata [19]

Az útvonal felderítését minden esetben azon fél kezdeményezi, amelyiknek közlendője van (azaz „on-demand" jellegű ad-hoc routing eljárást valósítottak meg a szerzők). Az említett fél a hirdetményeken keresztül közli szórásos jelleggel minden környező eszközzel a céleszköz címét. Azon eszközök, melyek vették eme hirdetményt, szintén ugyanezt a folyamatot hajtják végre, így a keresés az egész hálózatban szórásos jelleggel hajtódik végre.

Amint egy eszköz „magára ismer" a hirdetményekben, kapcsolatot létesít a hirdetmény forrásával, amely eszköz hasonlóan tesz a korábbi hirdetmény forrásával, és így tovább, egészen a keresés forrásáig. Ennek eredményeképpen már rendelkezésre is áll egy olyan multihop útvonal, melyen az adatok átvihetők.

Megjegyzendő, hogy bár a megoldás az általam eddig leírtaknál valamelyest kifinomultabb, mégis annak jellegéből fakadóan a hálózatot részleges ismeretét (végpontok) mindenképpen igényli (ami az új eszközök rendszerbe való integrálását valamelyest nehezebbé teszik). Ezen a felül a GATT procedúrák (pl. szolgáltatás felderítés egy „távoli" eszközön) csak igen körülményesen valósíthatók meg. Ennek legfőbb oka a Bluetooth Low Energy IC-k gyártóinak jelenlegi alkalmazásfejlesztési szemlélete, miszerint a BLE stack, bár ingyenesen elérhető a megfelelő API-n keresztül a SoC (System-on-a-Chip) rendszereken, azok belső függvényei és struktúrái (melyek a csomagok összeállításához és értelmezéséhez szükségesek) részben, vagy teljes

egészükben rejtve maradnak az alkalmazásfejlesztők előtt [20] [21]. Így, amennyiben ténylegesen teljes GATT (vagy azzal közel azonos) funkcióhalmazt szeretnénk ilyen jellegű multihop linkeken elérhetővé tenni, úgy gyakorlatilag egy kisebb BLE stack újraírására kényszerülünk. Csak akkor valósíthatjuk meg ezen szolgáltatásokat, ha nyílt vagy saját fejlesztésű BLE stacket alkalmazunk [22].

Mindez összességében azt eredményezi, hogy a megoldás csak egy adott szolgáltatáshalmaz kontextusában értelmezhető (kiegészítő szolgáltatásként, olyan esetekben, amikor a gyártónak kifejezett igénye van erre és az adott rendszer telepítését támogatja egy külön folyamat, mely a végpontok címeit kezeli), így korántsem tekinthető általános, valamint transzparens megoldásnak.

2.2. Scatternet linkek bevezetése az alsóbb rétegekben

Bár a szabványban semmilyen megjegyzés nem utal az alpont címében említettek irányába (sőt a hagyományos Bluetooth technológiában a scatternet topológia közel 10 éve támogatott, mégsem kerültek szabványosításra a multihop linkek [2]), mégis egy amolyan „gondolatkísérlet” erejéig tegyük fel, hogy mégis bevezetésre kerülnének ilyen kapcsolatok az L2CAP és Link Layer (MAC) rétegekben.

Ekkor, bár nem szabadulunk a végpont ismeretének peremfeltételétől, a stack MAC rétege képes lehet végrehajtani a szükséges procedúrákat, valamint az L2CAP számára biztosítani egy olyan „handle” objektumot, amelyen keresztül elérhetjük a távoli eszközt. Ekkor az egyes GATT specifikus funkciók az Attribute Protocol segítségével transzparens módon végrehajthatók a felsőbb rétegekből.

A kérdés ekkor azonban az, hogy milyen ad-hoc routing eljárásokat (on-demand, tábla-alapú, stb.) alkalmaznak ezen rétegek, az milyen alkalmazás-specifikus igények esetén lehet az optimális, valamint, hogy érdemes-e egyáltalán ekkora overheadet a stack-be ágyazni? A másik lehetőség ebből a szempontból nézve, hogy az alkalmazási rétegre bízzák ezen lehetőség megvalósítását, és nem definiálnak semmilyen eljárást erre a feladatra, csak a szükséges MAC rétegbeli funkciókat biztosítják az alkalmazáshoz, elkerülvén ezzel a stack „felesleges” divergálódását. Feltehetően ezért nem kerültek a szabvány szintjén kidolgozásra ezen megoldások a hagyományos Bluetooth technológia funkcióhalmazában, valamint ebből kifolyólag feltehetően a Bluetooth LE-ben sem fog ebbe az irányba evolválódni a specifikáció (ugyanakkor ez természetesen nem zárható ki).

II. Named Data Networking

A Named Data Networking (NDN) egy olyan Future Internet architektúra, melynek gyökerei egészen 2006-ig nyúlnak vissza, amikor is Van Jacobson a Google Tech Talk rendezvényén először ismertette az CCN (Content Centric Networking) alapkoncepcióját [23]. Az előadás során kiemelésre került a jelenlegi TCP/IP-alapú hoszt-centrikus architektúra korlátozottsága egy olyan hálózatban, melyet alapvetően információterjesztésre és -elérésre használunk, emellett bemutatásra került az CCN adatcentrikus architektúrája, mely már a tervezéskor sem a jelenleg is zömében alkalmazott „párbeszédjellegű" kommunikációt (ahol a hálózat legfőbb feladata a két végpont közötti kapcsolat biztosítása), s így az adatok helyét vette alapul, hanem magukra az adatokra helyezte a hangsúlyt.

2007-ben a PARC (Palo Alto Research Center) CCN (Content Centric Network) elnevezéssel indította útjára azon projektet, melynek célja a fentebb leírt koncepció (akkori formájában inkább csak ötlet) kidolgozása és implementálása volt. 2009-ben meg is jelent az első publikus specifikáció és open-source implementáció [24].

2010-ben az Egyesült Államokban az NSF (National Science Foundation) finanszírozta a NDN (Named Data Networking) projektet, mely azóta is folyamatban van [25]. Mindeközben 2011 környékén (az első „toborzó" előadás ekkorra datálódik [26]) az IRTF (Internet Research Task Group) is kiemelt figyelmet fordított a koncepcióra minek során megalapította az ICN RG-t (Information-Centric Network Research Group), mely szintén ezen architektúra kidolgozásán munkálkodik [27]. Megjegyzendő, hogy az említett meghatározó projekteken kívül több európai kutatási és fejlesztési tevékenység is indult [25].

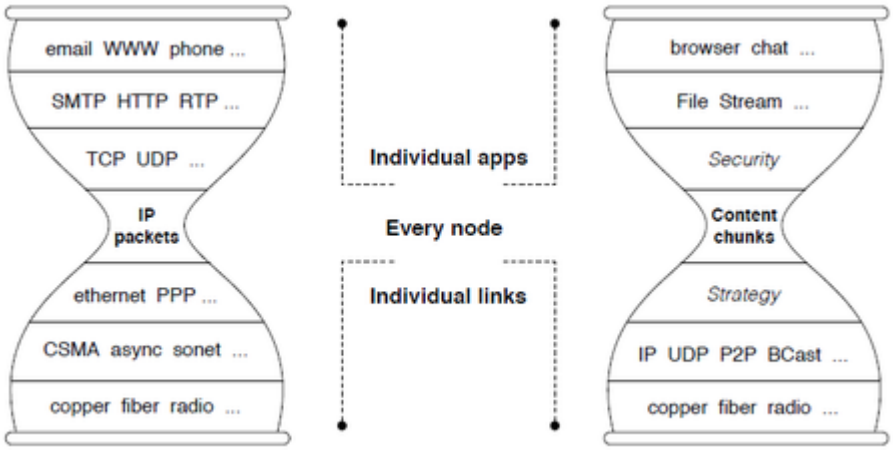
Látható, hogy bár az elnevezés korántsem sztenderdizált (NDN, ICN, CCN), mégis mind ugyanazon koncepciót jelöli. Mindez természetesen azt is magában hordozza, hogy nincs jelenleg végleges, avagy szabványos NDN hálózat, specifikáció, vagy architektúra, a kutatások jelenleg is töretlenül folynak.

A következő alpontokban előbb ismertetem a végletekig leegyszerűsített koncepciót, majd bemutatom az eddig kidolgozott architektúrát, mely ezt megvalósítja, végül röviden tárgyalom a megközelítés jelenlegi korlátait és főbb kihívásait.

1. Alapkoncepció

A koncepció lényege (ahogyan az már korábban is említésre került), hogy az egyes állomások helyett magukat, a számunkra érdekes adatokat nevezzük meg egy alkalmas URI (Uniform Resource Identifier, pl.: …/wikipedia/ndn/architecture/) segítségével. Ezen neveket egy adott csomagba ágyazva elküldjük a hálózat felé, amely a név alapján irányított keresést/felderítést végezve visszatér magával az adattal [28].

A megoldás, bár első ránézésre igencsak hasonlít a DNS (Domain Name System) alapfunkciójára, annál azonban sokkalta több, hiszen nem állomáscímet kapunk vissza, amelyen újra próbálkozhatunk, hanem magát az adatot. Az alapelv inkább analóg egy (ha az internetet vesszük alapul, akkor igen nagyméretű) elosztott adatbázisban való kereséssel, avagy lekérdezéssel, melyben érvényesül azon elv, hogy nem kell feltétlenül tudnunk, hogy az adat igazából melyik eszközön (virtuális vagy valós) található.

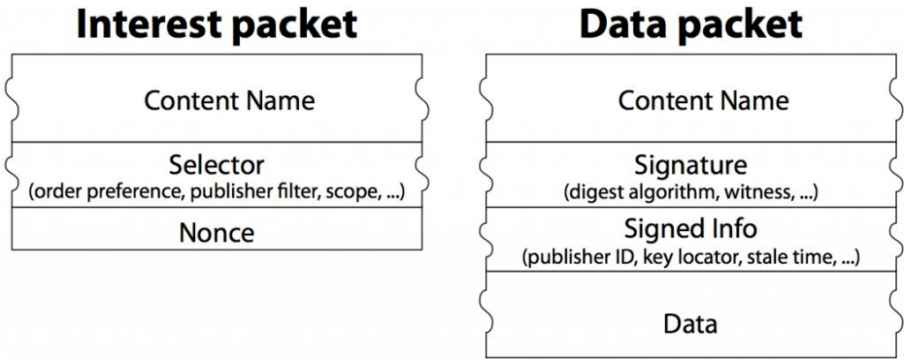


15. ábra – Az IP homokóra modell és az NDN architektúra viszonya [28]

Bár az alkalmazási szintből szemlélve ez nem sokban különbözik a jelenlegi rutintól, a különbség az, hogy mindez az OSI (Open Systems Interconnection) és NGN (New Generation Network) modellek hálózati szintjén is pontosan ugyanígy történik, ami összességében nagymértékben leegyszerűsíti a hálózati alkalmazások fejlesztését és működését, hiszen a legtöbb esetben a köztes rétegek (melyek részben vagy egészben az IP „csatornajellegének” következtében születtek) funkcionalitására immáron nem lesz szükség.

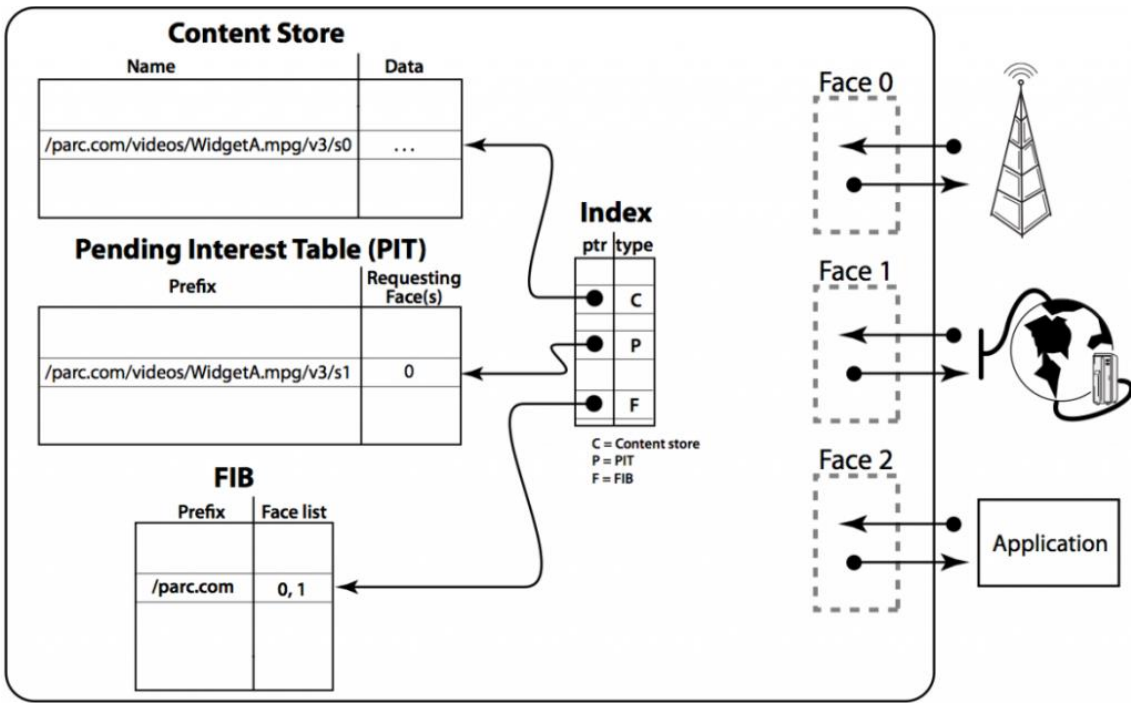
2. Architektúra

Az architektúra mindösszesen két csomagtípust különböztet meg. Az első az „Interest” csomag, mely az érdeklődésünk tárgyának (az URI-val jelölt adat) kifejezését teszi lehetővé, a másik a „Data” csomag, amely egyfajta „visszatérési értékként” magát az adatot hordozza [28].



16. ábra – Interest és Data csomagok tartalma [28]

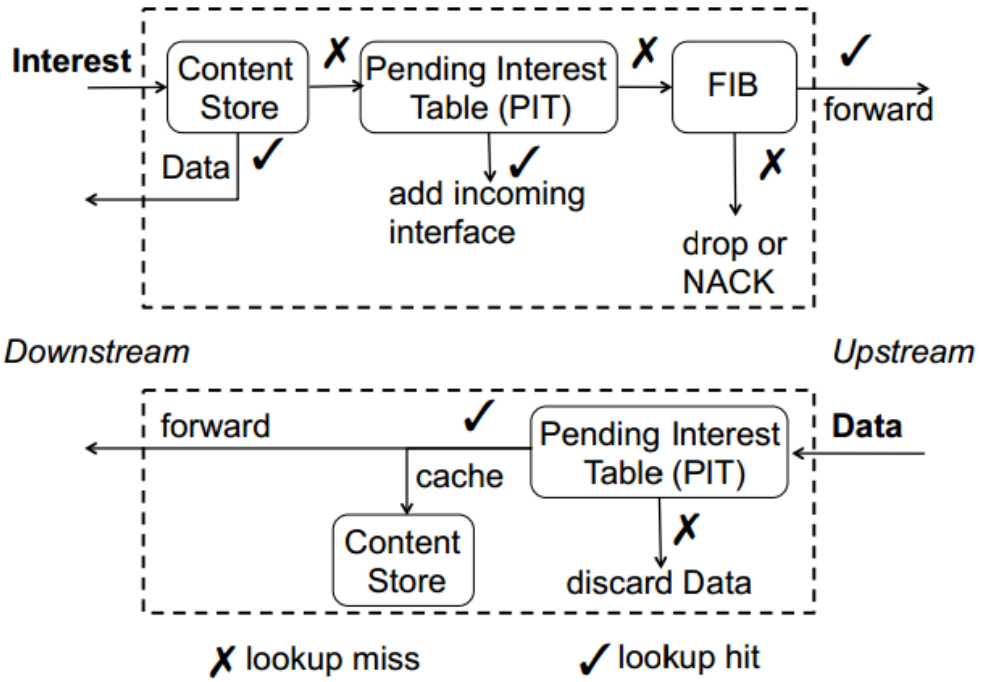
Az egyes csomagokkal történő különböző hálózati operációkat az NDN Node-ok végzik, amelyek sematikus felépítését a következő ábra szemlélteti.



17. ábra – Egy NDN Node felépítése [28]

2.1. Csomagtovábbítás

A kommunikációt az NDN hálózatokban minden esetben az adat iránt érdeklődő fél kezdeményezi egy alkalmas Interest csomag összeállításával (alapfeltétel jelen esetben az adat nevének ismerete – és hierarchikus szerkezete -, melyet a Content Name mezőbe írunk be), majd elküldésével egy NDN node irányába.



18. ábra – A csomagok továbbításának folyamata [29]

Az NDN node először felderíti a Content-Store tartalmát, és amennyiben teljes egyezést talál, úgy visszaküldi a tartalmat egy Data csomag formájában. Amennyiben nem talált, feljegyzi a bejövő Interest csomag interfészét és az adat nevének adott részét (ha talál egyezést, akkor csak feljegyez egy újabb interfészt, a már aktuális név mellé) a PIT-be (Pending Interest Table). Ezt követően a FIB (Forward Information Base) bejegyzései alapján a „Longest Matching Rule” elvet követve továbbítja egy alkalmas interfészen a keresztül ugyanazt az Interest csomagot, vagy amennyiben nem talált egyezést eldobja. (megj.: amennyiben nem talál egyezést, úgy alkalmazható lehet a „split-horizon” elv mentén történő szórásos továbbítás, vagy egy default útvonalat is megadhatnánk).

Amennyiben adatcsomag érkezik, és van egyezésünk a név tekintetében a PIT bejegyzései között, úgy beírjuk a Content Store-ba, és továbbítjuk a korábban feljegyzett Interest csomag bejövő interfészén az adatot. A Data csomag kiküldését

követően töröljük a bejegyzést a PIT táblából (valamint adott esetben frissítjük a FIB tartalmát az új prefiummal).

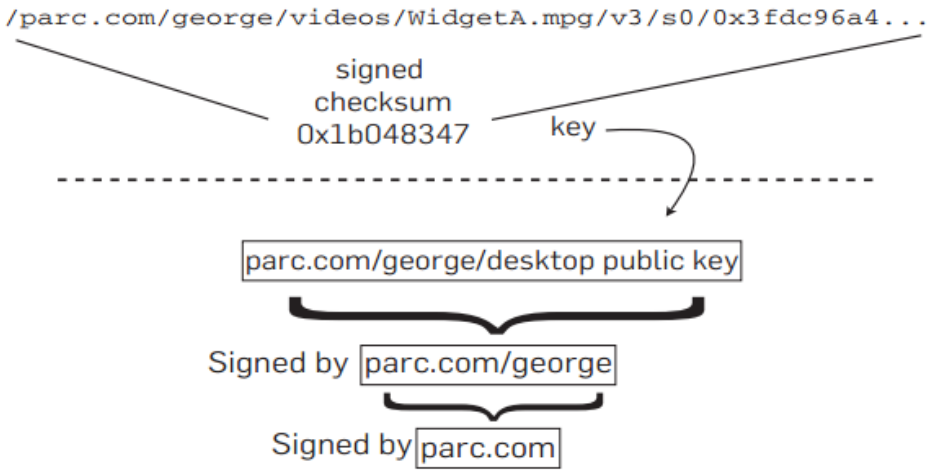
Láthatóan ezen igen egyszerű működéssel többes ugrásos csomagtovábbítást tettünk lehetővé, bármilyen a hálózatra vonatkozó információ nélkülözésével. Mindemellett számos olyan probléma alapját sikerült megszűntetnünk, melyek az IP csatorna-alapú megközelítéséből fakadnak. Ilyen például a broadcast és multicast csomagtovábbítás, melynek tényleges megvalósítása jelenleg is kihívás az IPv6-alapú hálózatok világában. Jelen esetben, ha ugyanarra az adatra érkezik több érdeklődés, és már él egy érdeklődés PIT-ben, úgy nem kerül az Interest továbbításra (csak a Data). Ez a folyamat tekinthető egyfajta feliratkozásnak is egy adatra, mely a módszer jellegéből fakadóan implicit módon támogatott. Inverz logikát követve a hagyományos értelemben vett DoS támadások is értelmét veszítik [30].

Önmagában a „split-horizon” elv mentén történő továbbítás nem garantálja a hurkok kialakulásának elkerülését. Erre szolgál az Interest csomagban a Nonce mező, mely egy 4 bájt hosszú véletlen szám, és melyet minden esetben az adat iránt érdeklődő fél generál. Ezzel azonosíthatók a már feldolgozott Interestek, így eldobhatók a hurokba került csomagok. (Megj.: ugyanakkor egy visszajutó csomag egyben információt is szolgáltat egy alternatív útvonal jelenlétéről, mely tipikusan linkhiba esetén jó szolgálatot tesz, sőt egyben lehetőséget teremt a hosszabb útvonalak kiszűrésére is egyszerű csomagszámláló statisztikával.) Mindemellett a jelenleg is alkalmazott routing protokollok (RIP, OSPF, IS-IS, BGP, stb.) többé-kevésbé minimális változtatásokkal képesek lehetnek NDN alapú hálózatokban operálni (az IP címek prefixumai helyett nevekét használjuk) [28].

További igen fontos következménye a megoldásnak, hogy egy célba érkező Interest, és az arra válaszképpen érkező Data csomag ugyanazon az útvonalon közlekedik a hálózat egészében, ami alapjában tér el az IP-alapú megoldásoktól, hiszen ott ez közel sem garantálható. Ennek és a Content Store jelenlétének eredményeképpen maga az adatsík is intelligenciával vértezhető fel, aminek következtében akár csomag/adat szinten is különböző procedúrákat alkalmazhatunk (pl. terheléselosztás, torlódásvezérlés, linkhiba detektálás, stb.).

2.2. Adatközpontú biztonság

Az NDN hálózatokban az adatcsomagok digitális aláírása kötelező (erre szolgálnak a Signature és a Signed Info mező). A Signature mező egyben ellenőrzi is a csomag integritását és validálja, hogy tényleg az adat és az elérési út került vele aláírásra. Mivel a publikus kulcs alapján történik a validáció, ezért a hálózat önmaga is ellenőrizheti az adatok hitelességét, így kizárhatja a nem hiteles csomagokat a hálózatból (célszerűen nem minden NDN node teszi ezt meg minden csomaggal). Fontos megjegyezni, hogy nincs szükség a kulcsok megjegyzésére a hálózat vagy a végpont által, hiszen az Interest csomagok Selector mezejében, valamint a Data csomagok Signature mezejében a Keyselector bejegyzés éppen az adat forrásának publikus kulcsát azonosítja [31], így azt egy újabb Interest segítségével bármikor lekérhetjük [32].



19. ábra – Hierarchikus „aláírásfák” [32]

Az aláírások ezen felül hierarchikusan szervezhetők, hiszen magát a kulcsot hordozó adatcsomagot is aláírhatja valaki más, és így tovább, ami lehetővé teszi a megbízhatóság csatolását (ha A-ban megbíztam és B eleme A-nak, akkor B-ben is megbízhatok), s így elkerülhető a felesleges vizsgálatok összessége. Az eddigiekhez hasonló elvet követve a csomagok, valamint a nevek parciális titkosítása is megvalósítható, a kulcsok pedig eloszthatók a hálózatban az eddig ismert eljárások segítségével (PKI, Diffie-Hellmann, stb.) [32].

Mindezen felül az állomásnevek hiányában az Interest forrása gyakorlatilag semmilyen módon nem azonosítható, azaz tökéletesen anonim módon történik az adatokhoz való hozzáférés (aminek akad számos előnye és hátránya is, melyekre most terjedelmi okokból fakadóan nem térek ki).

3. Kihívások

Bár az NDN önmagában számos problémára megoldást jelenthet, jó néhány kihívással is szolgál a kutatók számára. Az első és egyben legfontosabb a „Longest Matching Rule” alkalmazása egy olyan névtéren, mely jellegéből fakadóan elméletileg végtelen. Ezen felül a publikusan elérhető adatoknak egyértelmű nevekkel kell bírniuk, ami a névteret tovább növeli (ugyanakkor elviekben csökkenti a duplikált adatmennyiséget, hiszen ezen architektúra mentén egyszerűbb hivatkozni az adatokra, mint duplikáltan tárolni). Ez a rendszer skálázhatóságát erősen korlátozza.

További fontos probléma a skálázhatóság tekintetében a Content Store-ok és a cache-ing kérdésköre, mely (azon felül, hogy jogi aggályokat vet fel) a jövőbeni NDN node-ok (jelenleg routerek) tárhellyel való bővítését vetíti előre, aminek mérete, elosztottsága és optimális kihasználásának biztosítása jelenleg is erősen kutatott terület, ami szoros kapcsolatban áll a „forwarding strategy” kérdéskörével [28].

Mindezen felül azon megoldások, melyek az IP-alapú és az áramkörkapcsolt hálózatokban viszonylag egyszerűen megvalósíthatók (hang és/vagy video streamek), azok jelen esetben jól definiált és determinisztikus névgenerációt (pl.: /adat/formátum/időszelet) igényelnek a forrástól. Ehhez szorosan kapcsolódó probléma a különböző QoS (Quality of Service) szintek biztosítása a hálózatban, ami szintén kutatott terület jelenleg [28].

Látható tehát, hogy bár az NDN architektúra számos szempontból tekinthető előrelépésnek, annak szabványszinten való kidolgozása úgy, hogy általános jelleggel alkalmazható legyen, közel sem triviális feladat.

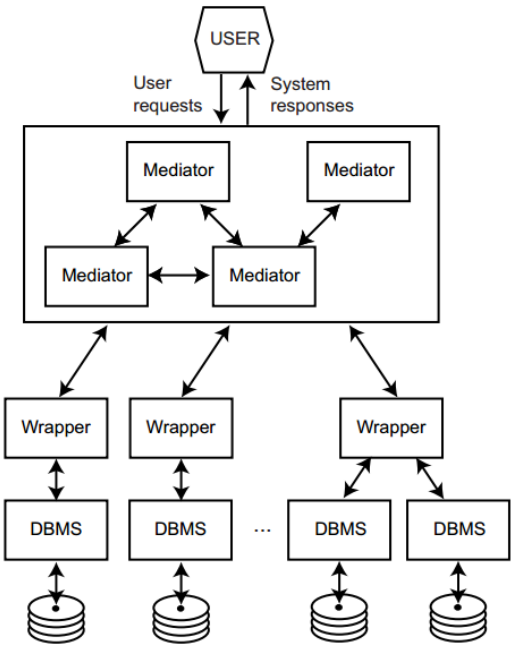
III. Szolgáltatás mediáció

Jelen pontban bemutatásra kerül az általam tervezett megoldás a GATT struktúrák és szolgáltatások multihop Bluetooth LE hálózatokban való kezelésére. Az első pontban ismertetem azt a nézőpontot, ahonnan szemlélve a Bluetooth Low Energy technológiát, az NDN koncepció alkalmazásának lehetősége igen jól látható. Azt követően ismertetem magát a tervezett GATT szolgáltatást, melyben kitérek az átültetett NDN architektúra-elemekre, ismertetem az ahhoz tartozó különböző eljárásokat, valamint végül kitérek a megoldás skálázhatóságára és a biztonság kérdésére is.

Megjegyzendő, hogy a megoldás korántsem tekinthető teljesnek, sokkal inkább „Proof of Concept” jellegű, melyben csak a kritikusabb funkciók kerültek kidolgozásra. Ugyanakkor látható lesz a későbbiekben, hogy a jelenlegi megoldás megfelelő pontokon való kisebb bővítéseivel az közel teljessé tehető.

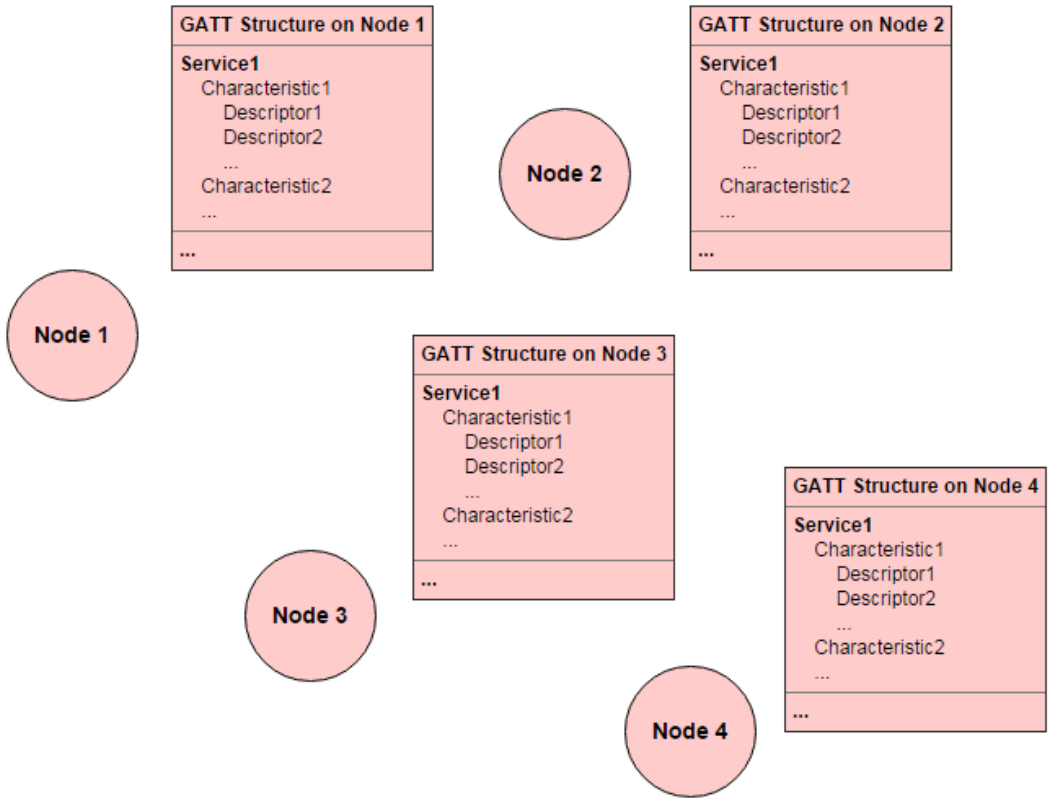
1. GATT struktúrák összessége, mint elosztott adatbázis

Az elosztott adatbázisok létrehozása és kezelése közel sem triviális feladat, hiszen számos megközelítés és architektúra létezik, amelyekkel megvalósítható lehet a partikuláris adatbázisok logikai összefűzése és az egységes nézet létrehozása, mellyel elfedhetjük az elosztottságot, ezzel egyszerűbbé téve a lekérdezések és tranzakciók mikéntjét (vegyük észre, hogy alapvetően az egyik fő cél itt is az adatok a tárolás helyétől való függetlenítése a felsőbb rétegek számára, csak úgy, mint az NDN hálózatokban) [33].



20. ábra – Mediator/Wrapper architektúra [33]

Meglévő adatbázisok összefűzésére és kezelésére az egyik leggyakrabban használt megvalósítási modell a mediator/wrapper architektúra. Ezen architektúrában a „wrapper" modulok olyan adapterfunkciókat (az adatbázisok esetleges heterogenitásából fakadóan) látnak el, amelyek segítségével a „mediator" modulok az egyes illesztett adatbázisok és a felhasználó közötti interakciót egyetlen, jól definiált interfészen keresztül megvalósíthatja (számos esetben csak fordítójelleggel vannak jelen a rendszerben, és a tényleges tranzakciókat maguk a wrapperek végzik). A mediátorok ezen felül hierarchikusan, vagy szövevényesen is szervezhetők (függően az alkalmazás igényeitől) [33].



21. ábra – kisebb GATT adatbázisok az egyes Bluetooth LE node-okon

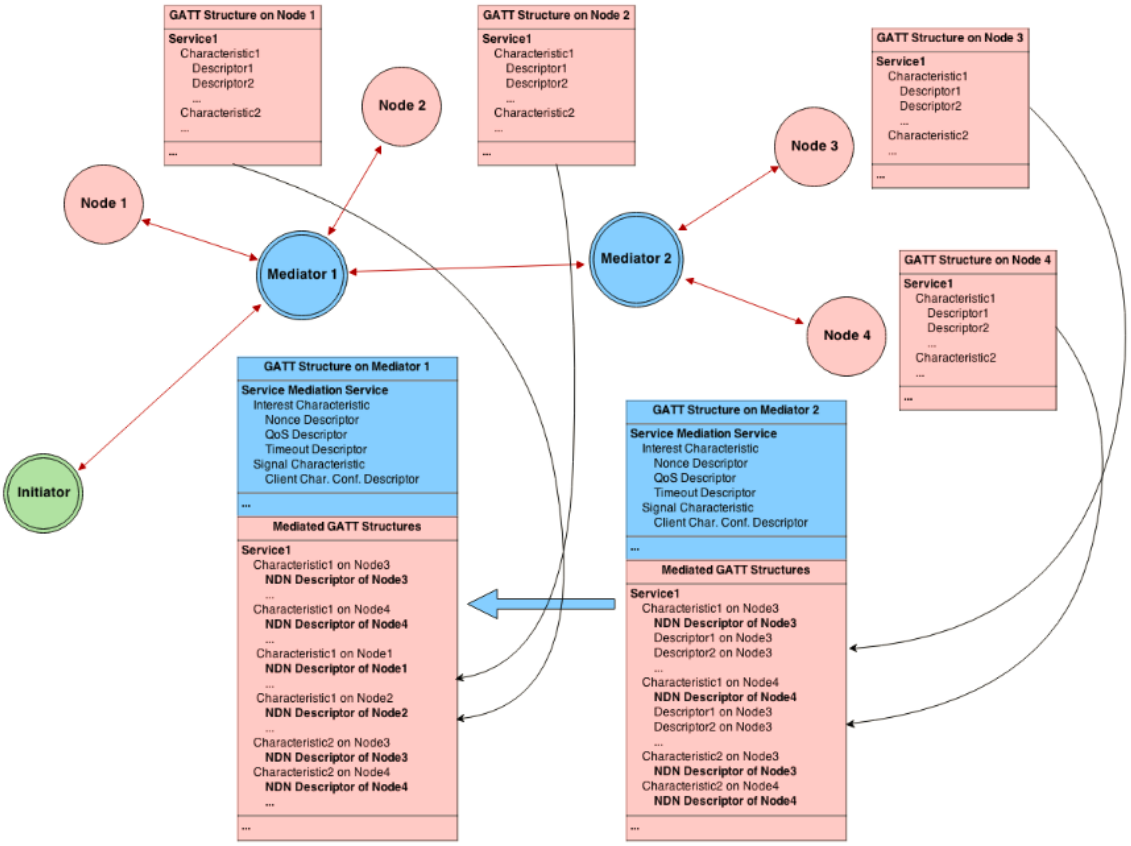
Amennyiben feltesszük, hogy tekinthetünk a Bluetooth LE eszközök összességére úgy, mint egy elosztott adatbázisra, és a különböző Characteristic értékekre úgy, mint magukra az adatokra, akkor egy tetszőleges bonyolultságú és kiterjedésű Bluetooth LE hálózatban a GATT specifikus funkciók (írás, olvasás, notifikáció, stb.) végrehajtása összességében nem jelent mást, mint ugyanezen funkciók végrehajtása egy elosztott adatbázisban.

Ugyanakkor mindehhez szükségünk van egy olyan nézetre, melyben láthatjuk, hogy milyen szolgáltatások, és azokon belül is milyen Characteristic és Descriptor értékek csatolódnak (ezen felül azt sem árt tudni, hogy melyik eszközön találhatók, ám jelen esetben ez egyáltalán nem kritérium). Tekintettel arra, hogy a GATT keretrendszer igen gondosan specifikálja a különböző szolgáltatások formátumait, így az adatbázisok homogenitásából fakadóan a „wrapper” modulokra nincs is szükség. Mindennek megfelelően csak olyan mediátor eszközök (vagy átvitt értelemben szolgáltatások) jelenléte szükséges, melyek képesek az egyes lekérdezéseinket egyértelműen kezelni, és ezek alapján az általunk kívánt nézetet kialakítani. Ezt valósítja meg a következő pontban taglalt „szolgáltatás mediációs szolgáltatás”.

2. Szolgáltatás mediációs szolgáltatás (SMS)

Az általam tervezett szolgáltatásban nézetek kialakítását minden esetben egy Initiator eszköz kezdeményezi, egy alkalmas karakterlánc (a következő alpontban részletesebben kifejtem) összeállításával, melynek értelmezése közel azonos az NDN architektúrában az adatok neveivel. Ezen felül a szolgáltatás egyes paraméterei (QoS, Timeout, Nonce) is beállítandók az Initiator eszköz által, a karakterlánc átküldését megelőzően.

A karakterlánc elemzésével és továbbításával a mediátorok hozzácsatolják az általuk látott eszközök szolgáltatásait a magukéhoz (ebből következően nem kell kitüntetett mediációs céleszköznek lennie a mediátornak, lehet egy egyszerű hőmérséklet szenzor, vagy egy „okos” hálózati aljzat is), majd a jelzésüzenetek segítségével, és további szolgáltatás aggregációkkal az Iniator-tól egy ugrásra levő mediátorokon kialakul egy-egy olyan nézet, mely a rajtuk keresztül elérhető hálózat szolgáltatás sokaságát reprezentálja.



22. ábra – Teljes nézet kialakítása a mediátorok segítségével

Az egyes Characteristic értékekhez tartozó GATT funkciók ezután transzparens módon végrehajthatók (kivételt képez ez alól a szolgáltatások felderítése, ugyanis azt csak mediátorokon keresztül végezhetjük el), úgy mintha ténylegesen magukon az eszközökön tennénk mindezt. A továbbiakban a működés részleteit taglalom.

2.1. Az NDN architektúra átültetése

Jelen pontban az egyes NDN architektúra-elemek (nevek, PIT, FIB, Content Store) ezen hálózatban való interpretációját taglalom.

2.1.1. Névtér

A Named Data Newtorking koncepció egyik sarkalatos pontja azon névtér, melyen az adatok összességét értelmezni tudjuk. Emellett nyilvánvalóan az egyértelműség feltételének is meg kell felelnünk, hiszen más esetben nem feltétlenül kapnánk azt az adatot, amelyikre alapvetően számítunk.

Szerencsére a Bluetooth LE GATT keretrendszeréből származtatható névtér közel sem végtelen, valamint egyetlen plusz információ segítségével egyértelművé is tehető a következő formátumnak megfelelően: „**/location/service/characteristic**”.

A „location” mező ez esetben az eszközt azonosítja, azonban nem feltétlenül szükséges Bluetooth MAC-címek alkalmazása (ugyanakkor kétségkívül a címek mindenképpen egyértelműek), alkalmazhatjuk ugyanis az eszközök nevét is (amennyiben ezek egyértelműek), amelyet a készülékek a hirdetményekben (Advertising event) kisugároznak, így a kapcsolódást megelőzően a BLE eszközfelderítése során ugyanolyan azonosítóként szolgálhatnak, mint maguk a MAC címek.

A „service” azon szolgáltatást azonosítja, amely érdeklődésünk tárgyát képezi. A Bluetooth SIG egyértelműen meghatározza a Bluetooth Assigned Numbers dokumentumában az egyes szolgáltatások UUID azonosítóit, ezen felül azok formátumait is (16, valamint 128 bites számok alkalmazhatók) [34]. A „characteristic” karakterlánc a szolgáltatásokkal ekvivalens módon azonosítható.

Bár a hierarchia legalsó szintjén helyezkednek el a descriptorok, azok inkább segítik az adatok (Characteristic) interpretációját (és segítik a különböző GATT funkciók végrehajtását), mintsem ténylegesen hordozzák azokat, így bár a névtérnek nem képezik részét, azok összessége becsatolódik a kialakítandó nézetbe, így a teljes felderítést követően elérhetővé válnak.

Noha az adatok nevének egyértelműnek kell lennie, maga a felderítés kifejezetten előnyös, ha nem feltétlenül az. Ennek megfelelően az egyes határoló „/” karakterek között a helyet akár üresen is hagyhatjuk. Így például, ha egy teljes felderítést szeretnénk végezni, elegendő egy „///” karaktersort összeállítani.

2.1.2. Forward Information Base (FIB)

Bár az architektúra igen egyszerűen értelmezhető objektumokat alkalmaz, melyek kiválóan implementálhatók a megfelelő környezetekben, jelen megvalósítás során figyelembe vettem, hogy jellemzően beágyazott rendszerekben kerülnek implementálásra ezen funkciók (ahol a RAM mérete igencsak korlátos, jellemzően 8–16KB, melynek tekintélyes részét a BLE stack foglalja le). Így memóriatakarékossági okokból, a FIB jelen megközelítésben csak egy olyan mutatókat tartalmazó tömb, mely az eddig felderített eszközök közül a mediátor funkciókkal rendelkezők címeire mutat (minthogy ezen tábla az NDN architektúrában az Interestek továbbítására szolgál alapvetően).

Az egyértelmű kapcsolatot egy lokálisan regisztrált, közvetített szolgáltatás Characteristic értékei, és a tőlünk egy ugrásra levő példánya között a lokális handle érték, a külső handle érték és a távoli (eredeti) eszköz MAC címének hármasa valósítja meg. Ez utóbbi a következő pontban ismertetett közvetített szolgáltatás regisztrációja során egy NDN Descriptor formájában kerül eltárolásra, míg a külső handle érték egy lokális-külső handle értékpárokat tartalmazó struktúrában kerül rögzítésre (bár alapvetően ezen struktúrát is lehetne FIB részének tekinteni, az NDN terminológiájával való keveredés elkerülésének érdekében mégsem teszem).

Emellett jelen megvalósítás feszítőfa topológiát alakít ki, melyben így egyértelmű az egyes távoli eszközök Initiator irányába küldött adatainak továbbítási módja (minden esetben a Central eszköz irányába), ezen felül az írás és olvasás továbbításának módja is egyértelműen meghatározható az előbbi bekezdésben felsorolt paraméterek segítségével.

Az ad-hoc routing tekintetében jelen megoldás is „on-demand” jellegű, azonban a QoS paraméter (III/2.2-es pont) segítségével bármikor konfigurálhatjuk úgy a mediátorokat, hogy „tábla-alapú” eljárást alkalmazzanak az adott lekérdezésben megjelölt nevekre. A név prefixumok külön (duplikált) tárolása azonban továbbra sem indokolt (hiszen elegendő lokális GATT adatbázis bejegyzéseit frissíteni, amelyekből automatikusan generálhatók az előtagok).

2.1.3. Pending Interest Table (PIT)

A Pending Interest Table jelen megvalósításban a feszítőfa topológia miatt nem szükséges, hiszen csak egyetlen (Central) eszköz irányából fogadhatunk Interesteket.

2.1.4. Content Store

A Content Store a GATT struktúra jellegéből fakadóan szintén elosztott módon áll rendelkezésre, hiszen nem csak maguk a szolgáltatások kerülnek közvetítésre, hanem azokon belül a Characteristic objektumok értékei is kiolvasásra (amennyiben ez a funkció engedélyezett az eredeti eszközön) és tárolásra kerülnek. Mindennek megfelelően külön objektum definiálására ez esetben sincs szükség.

2.2. A definiált szolgáltatás

A definiált SMS (Service Mediation Service) szolgáltatás a következőképpen épül fel:

- Service Mediation Service
  - Included Services
    - …
  - Characteristics
    - Mediation Signal Characteristic
      - Client Characteristic Conf. Descriptor
    - Mediation Interest Characteristic
      - Interest Nonce Descriptor
      - Interest QoS Descriptor
      - Interest Timeout Descriptor

Az „Included Services” szakasz tartalmazza a közvetített szolgáltatások összességét másodlagos szolgáltatásként (így nem téveszti meg az esetlegesen ezen szolgáltatásra nem felkészített alkalmazásokat, melyek a GATT adatbázisban keresve automatikusan aktiválnak bizonyos funkciókat a környező eszközökön).

A „Mediation Signal Characteristic” az alapvető jelzésfunkciókért felelős (típusa szerint 8 bites integer). A Client Characteristic Configuration Descriptor (CCCD) segítségével Indication funkcióra „iratkozhatunk fel”, így az érték változásakor egy megbízhatóan küldött értesítést kapunk, a különböző folyamatok eredményességét illetően. Az egyszerű hibadetekció érdekében egy MEDIATION_ERROR_BASE értéket vezettem be, melyhez az adott hibakódokat hozzáadva a jelzésüzenet származtatható. Ennek megfelelően egyetlen feltételvizsgálat elegendő a hiba

detekciójára, majd több vizsgálaton keresztül kideríthető a hiba oka. Ezen felül két nem hiba jellegű jelzést definiáltam, melyek egyike (MEDIATION_READY) a szolgáltatás elérhetőségét jelzi, míg a másik (MEDIATION_IN_PROGRESS) egy aktuális felderítés folyamatát jelzi.

A „Mediation Interest Characteristic” (típusa szerint: ASCII karakterlánc) szolgál a mediátorok bemeneteként, melybe az érdeklődésünk tárgyát leíró név beírásra kerülhet.

A kapcsolódó „Interest QoS Descriptor” alapvetően azt szabályozza, hogy az egyes lekérdezések eredményeiként előálló GATT struktúrák milyen gyakorisággal frissüljenek az aktuális állapotra a hálózatban, valamint, hogy milyen hosszú ideig éljenek a bejegyzések a lokális GATT adatbázisokban. Minthogy ez szorosan összefügg az NDN Forwarding Strategy kérdéskörével, mely jellegéből fakadóan mélyebb analízist (és mindenekelőtt megfelelő modelleket) igényel, ezen paraméter jelen esetben nem kerül használatra.

A „Interest Nonce Descriptor” egy 32 bites integer, mely funkcióját tekintve teljesen megfelel az NDN architektúrában betöltött szerepének, amely nem más, mint a hurkok kialakulásának elkerülése. A feszítőfa topológiából fakadóan azonban ezen paraméter szintén értelmét veszíti (később a szövevényes topológiák esetén lehet hasznos).

Az „Interest Timeout Descriptor” érték (8 bites integer) azon szekundumban mért időtartam, melynek letelte előtt, ha nem végeztünk a felderítéssel, úgy megfelelő hibajelzéssel térünk vissza a folyamatból.

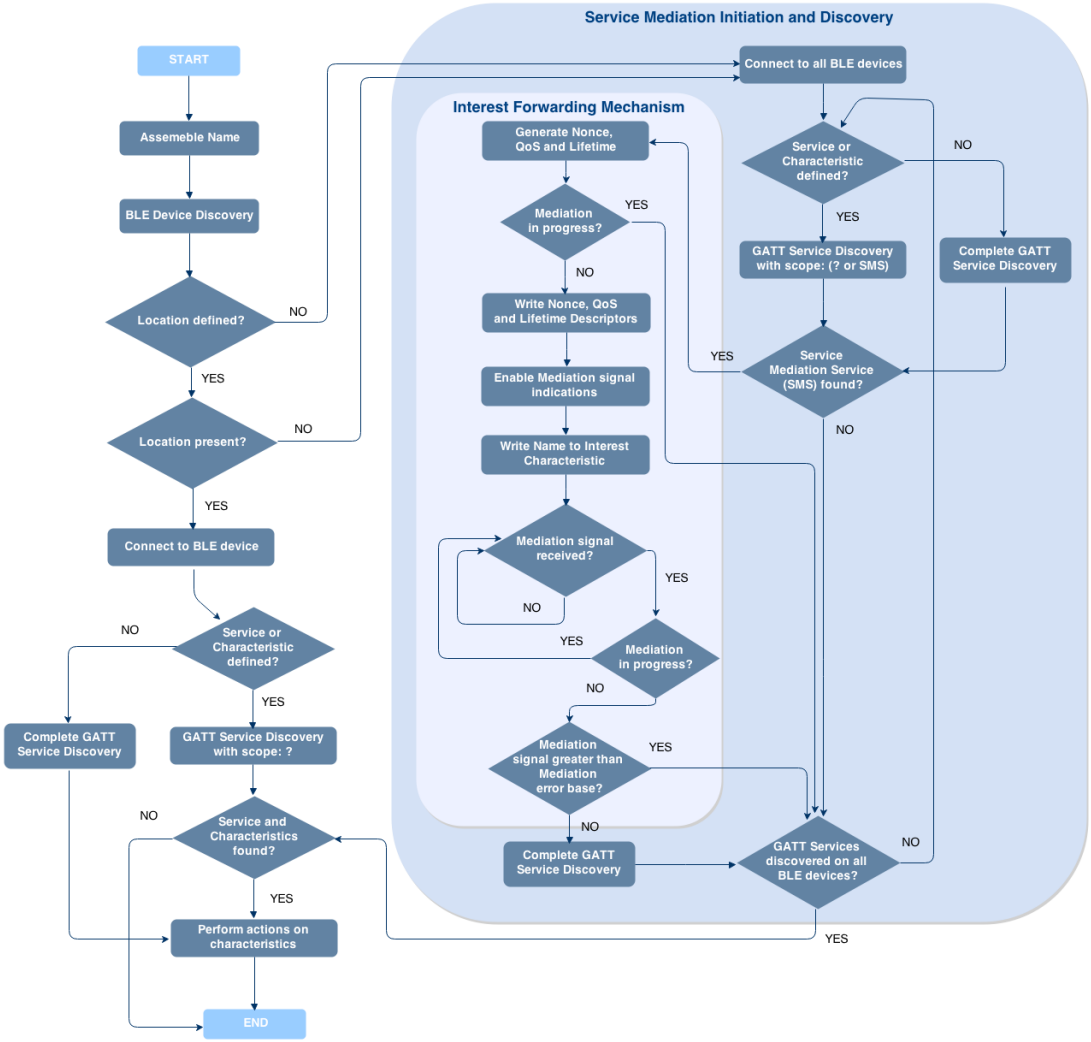
2.3. Fontosabb procedúrák ismertetése

Az alábbi pontokban ismertetem a szolgáltatás fontosabb folyamatait, úgy, mint az Initiator által elvárt működés a különböző nézetek kialakításának érdekében, a mediátorok elemzési és (al)nézetkialakítási metódusai, valamint a mediátorok által való GATT funkciók „átjátszásának” procedurái az Initiator szempontjából transzparens működés biztosításának érdekében.

2.3.1. Initiator által kezdeményezett felderítés

A szolgáltatás mediáció minden esetben egy alkalmas név összeállításával kezdődik, mely konform a korábban definiált formátummal. Bár ez a folyamat természetesen tekinthető egy felsőbb réteg feladatának is, jelen megoldásban ez az Initiator egység/modul feladata. A folyamat egy Bluetooth LE felderítéssel folytatódik,

mely a korábban leírtaknak megfelelően a hirdetési csatornákon való hallgatást foglalja magában, a megfelelő hirdetmények után. Az általánosságra törekedvén a következő folyamatábra azon eset bemutatását is magában foglalja, melyben nincs alapvetően szükség a mediációs folyamatokra (ekkor az adott szolgáltatás lokációja adott és az Initiator hatósugárban található).



23. ábra – Az Initiator működésének folyamata az SMS-ben

Amennyiben a lokáció ismeretlen, avagy nincs az Initiator hatósugarában, az kapcsolódik az összes környező eszközhöz és felderíti azok GATT struktúráit, az SMS (Service Mediation Service) és egyéb definiált szolgáltatások bejegyzései után kutatva. Minden olyan esetben, amikor talál SMS bejegyzést, kiolvassa a Mediation Signal Characteristic értékét és amennyiben MEDIATION_READY az érték, úgy feliratkozik a GATT Indication funkciójára. Ezt követően beírja a generált Nonce, QoS és Timeout értékeket a Descriptorokba, majd ugyanezt teszi a korábban összeállított névvel, beírja azt is az Interest mezőbe.

Erre a beírásra a Mediator egy Indicationt küld MEDIATION_IN_PROGRESS tartalommal. Az Initiator egészen addig vár, amíg a Mediator MEDIATION_READY „üzenettel”, vagy egyéb hibakóddal jelzi a mediáció befejeztét. Ezt követően egy teljes GATT felderítést végez az Initiator eszköz az adott Mediator eszközön, mely során hozzájut a közvetített szolgáltatások összességéhez. A folyamat akkor ér véget, ha minden környező eszközön végre tudtuk hajtani a szolgáltatások teljes felderítését (vagy hiba esetén átléptünk rajtuk).

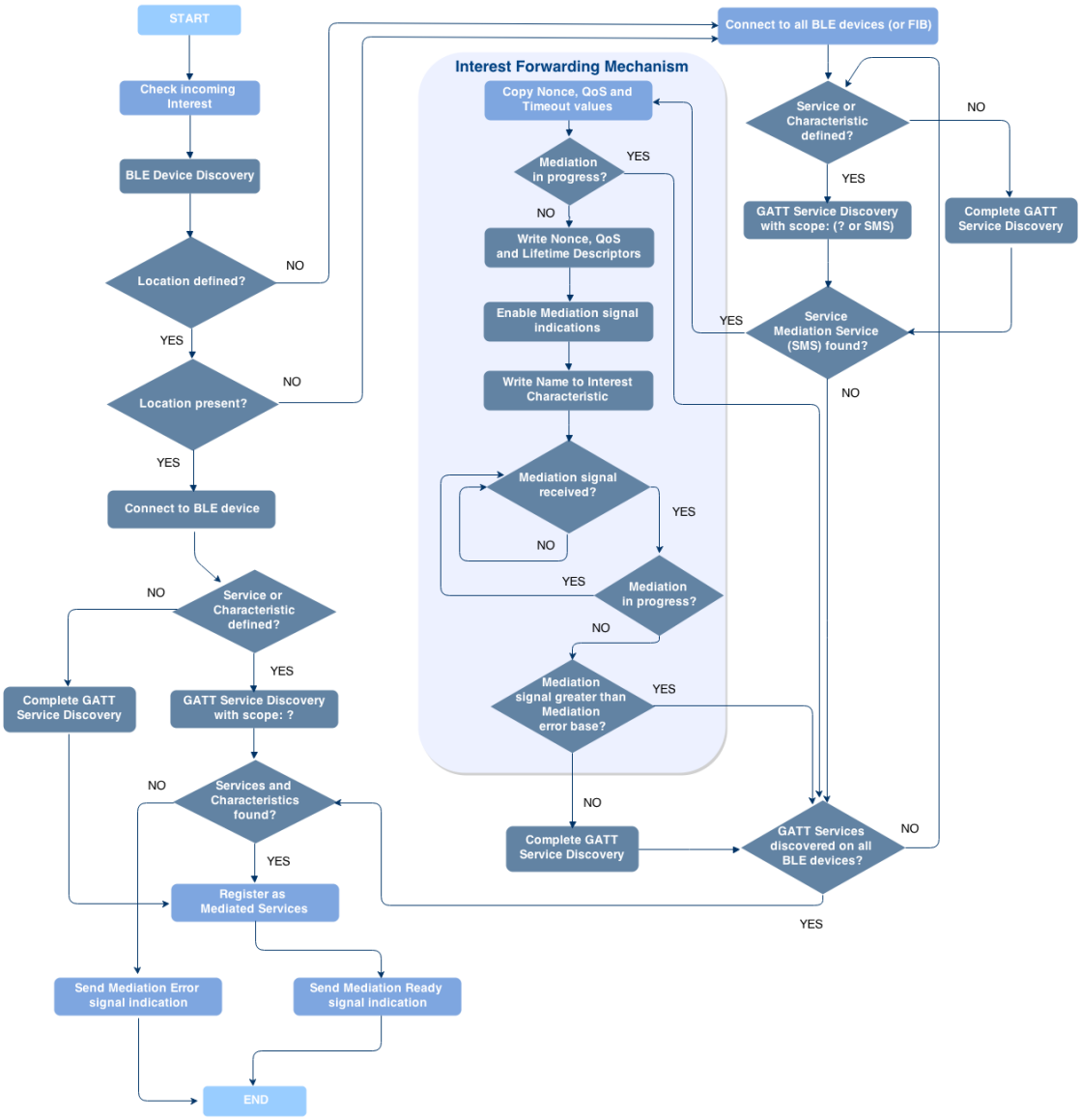
A folyamat természetesen részben párhuzamosítható, hiszen amíg egyik eszközön mediációs folyamat megy végbe, addig vizsgálhatunk további eszközöket, valamint átadhatjuk más eszközöknek az általunk generált Interesteket.

2.3.2. Interestek feldolgozása a mediátor eszközökön

Az Interestek feldolgozása és mediációs folyamat végrehajtása - az elvégzendő feladatok szintjén - közel azonos az Initiator működésével, ami nem meglepő, hiszen a mediáció során a hierarchiában egy alsóbb mediátor számára az eggyel felette levő mediátor Initiatorként „viselkedik”, ami biztosítja a transzparens működést (semelyik mediátor node nem tudja, hogy a hierarchia mely szintjén foglal helyet) és így az egyes eszközök tetszőleges hierarchiába való szervezését.

A folyamatot jelen esetben egy bejövő Interest triggereli, melynek vizsgálata során (tokenizálás) kinyerhetjük a működéshez szükséges információkat (location, service, characteristic). Amennyiben a megoldásba bevezetésre kerül a mesh topológia lehetősége, úgy ezen fázisban vizsgálhatjuk a Nonce értéket a hurkok elkerülésének érdekében.

A folyamat ezt követő szakasza lényegében megegyezik az előbbi pontban leírtakkal, azon különbséggel, hogy a mediátor nem generál új QoS, Nonce, Timeout és Interest értékeket, hanem a továbbítás során ezeket egyszerűen lemásolja. Megjegyzendő, hogy a Timeout érték kezelése a megfelelő elméleti modellek és mérések hiányában csak „hasraütés-szerűen” tehető meg, így a mediátor eszközökön, csak azok másolása történik meg, értelmezése azonban nem (csak az initiator egységen). Bár alapvetően el is hagyhatnánk a szükségtelen értékek beírását, mégis azok - a működés megfelelő karakterizálásának lehetőségét biztosítandó - átvitele mellett döntöttem, hiszen míg az értelmező logika alapvetően gyorsan lefut, addig a GATT üzenetváltásokra ez már nem teljesen igaz.



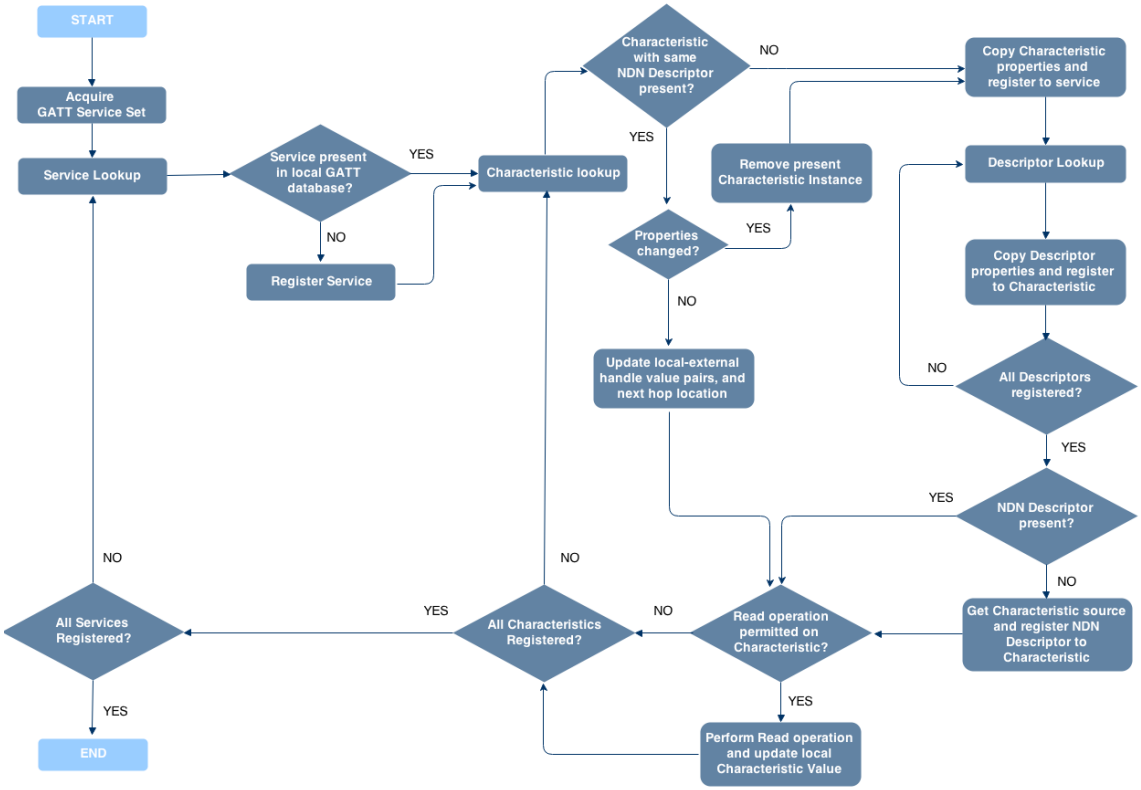
24. ábra – Interestek feldolgozása a Mediátor eszközökön

Szintén megjegyzendő, hogy a már meglévő közvetített GATT struktúrák vizsgálatát a felderítéskor jelen megoldás során nem végzik el a mediátor és initiator eszközök. Ennek legfőbb oka az ezen folyamatot szabályozó QoS paraméter kidolgozatlansága, minek következtében jelen megvalósítás során az initiator minden egyes nézetkialakítási kezdeményezése, a legaktuálisabb hálózati kép kialakításának érdekében egy, a teljes hálózatban való keresést fog eredményezni (hiszen változhat mind a topológia, mind a struktúrák összetétele időközben).

A GATT szolgáltatások felderítését követően vizsgálatra kerül, hogy egyáltalán sikerült-e bármilyen szolgáltatást felderíteni (üres halmaz vizsgálat). Amennyiben igen, úgy azok regisztrációra kerülnek a lokális adattárba, majd a megfelelő jelzésüzenetek generálódnak az initiator eszköz irányába.

2.3.3. Közvetítendő szolgáltatások regisztrációja a Mediátor eszközökön

A közvetítendő szolgáltatások regisztrációja összességében nem sokban különbözik egy általános szolgáltatás regisztrációjának folyamatától a GATT keretrendszerben. Ugyanakkor jelen megoldás során memóriatakarékosság és az átláthatóság érdekében a Characteristic értékek aggregáltan kerülnek regisztrálásra, azaz míg a szolgáltatásokból csak egy-egy példány kerül tárolásra (függetlenül attól, hogy melyik eszközön van jelen), addig a Characteristic-ekből több ugyanolyan UUID-val rendelkező is eltárolásra kerül (különböző NDN deszkriptorokkal). Ennek legfőbb oka, hogy az egyértelmű megkülönböztetés (és értéktárolás) csak így tehető meg.



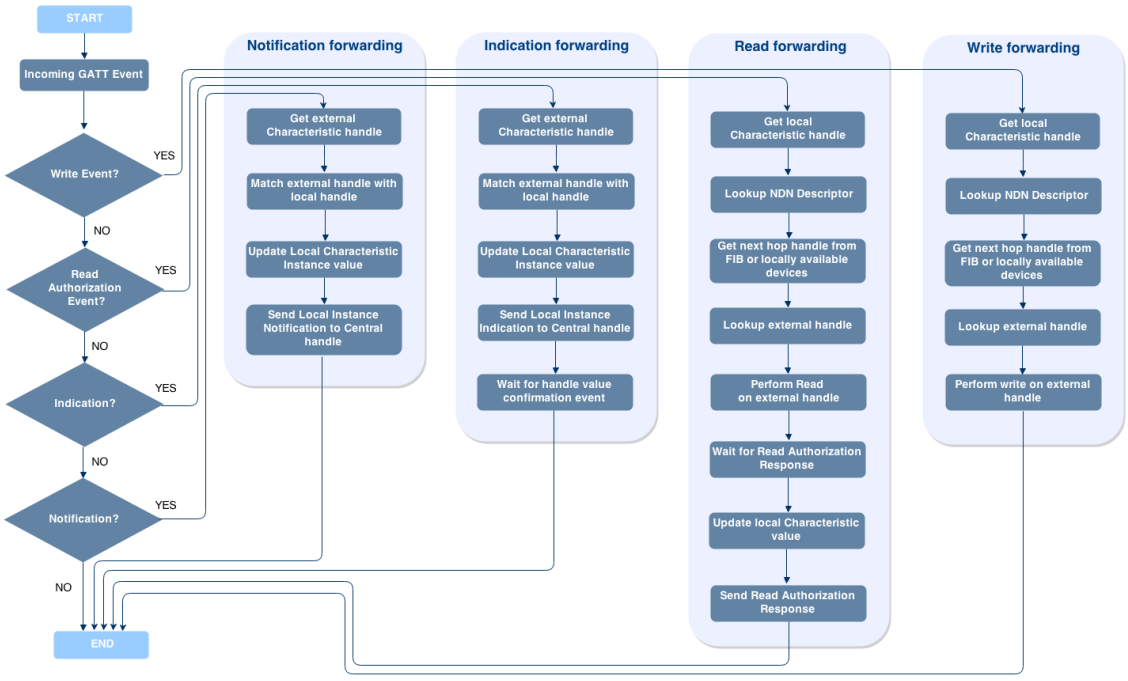
25. ábra – A közvetítendő szolgáltatások regisztrációjának folyamata

Megjegyzendő, hogy a megoldás jellegéből fakadóan előfordulhat, hogy a topológia változásából, vagy pusztán a nézetkialakítási kezdeményezések ezen rétegből észlelt nem-determinisztikusságából fakadóan ugyanazon Characteristic-ek (ugyanolyan NDN deszkriptorral) az adott GATT struktúra életciklusa alatt többször is felderítésre kerülnek, ami ha nem kezelnénk, akkor memóriaszivárgáshoz vezetne. Mindemellett a nézet egyértelműsége is megszűnne, hiszen nem tudnánk, hogy melyik bejegyzés az, amelyik ténylegesen az adott értékre mutat. Ebből fakadóan, ha már létezik a bejegyzés és tulajdonságai sem változtak, akkor egyszerűen frissítjük a továbbításhoz szükséges belső struktúrákat.

Fontos ezen kívül megjegyezni, hogy amennyiben a Characteristic tulajdonságai közé az olvashatóság hozzátartozik, úgy a regisztráció során frissítjük is (a GATT olvasási funkciójának segítségével) az aktuális értékkel a lokális példányt. Ez összességében azt jelenti, hogy a kiolvasás a hierarchia legalsó szintjétől indul és az átjátszás során folyamatosan átadódnak a frissen kiolvasott értékek, egészen a hierarchia legfelső szintjén elhelyezkedő eszközig. Ennek oka szintén a konzisztens nézet kialakításában keresendő, aminek jelen megközelítésben a tényleges adatok szintjén is meg kell lennie (azonban ez a QoS paraméterrel szintén szabályozható lehet). Emellett hasznos lehet olyan szituációkban, amikor az adat viszonylag statikus és nem szeretnénk feltétlenül egy friss nézetet kialakítani annak kiolvasásának érdekében (pl.: Device Information Service [35]).

2.3.4. GATT funkciók átjátszása a Mediátor eszközökön

Amint a mediátorok mindegyike visszajelzett az initiator irányába az aktuális nézetnek megfelelően, az initiator egység végrehajthatja a különböző GATT specifikus funkciókat (a GATT szerveroldalán ezek aszinkron eseményekként jelennek meg), amelyek átjátszását a mediátorok a következő ábrán látható módon valósítják meg.



26. ábra – BLE GATT események kezelése a mediátorokon belül

Jóllehet az ábra viszonylag sematikus, láthatóan igen egyszerűen végrehajthatók az egyes Notification és Indication események „továbbítása” a megfelelő irányba, tekintettel a feszítőfa topológiára. Az írási művelet továbbítása ugyan egy fokkal bonyolultabb, hiszen a továbbítás során el kell döntenünk, hogy milyen irányba tesszük

mindezt. Megjegyzendő, hogy az írás bár ezen szintből szemlélve „best-effort” jellegűnek hat, a GATT ezt a folyamatot többféleképpen is képes végrehajtani. Ezen módok a Characteristic tulajdonságainál kerülnek definiálására az eredeti eszközön (pl.: write without response, write, authorized write, stb.). Minthogy ezeket a regisztráció során egyszerűen lemásoltuk, az eredeti definíció determinálja a mediációs hálózat egészében a viselkedést.

Az olvasás művelete jelentette igazából a legnagyobb kihívást, hiszen alapesetben erről semmilyen értesítést nem kapunk a különböző BLE stackekből. Azonban létezik egy „read authorization request” (Volume 3: Part G – 8. Security Considerations [17]) esemény, melyet akkor kapunk, ha a Characteristic metaadatait annak regisztrálásakor ennek megfelelően állítjuk be. Ennek segítségével az olvasási esemény is észlelhetővé válik, azonban kikötés jelen esetben, hogy az egyszerű olvashatóság helyett a közvetített szolgáltatások regisztrációjánál ezen paramétert kell beállítani (amennyiben olvasható a Characteristic). A kérés válaszaként „read authorization response” üzenetet küldünk, amint ugyanezt a választ megkaptuk a szomszédos mediátortól, avagy az olvasást sikeresen végrehajtottuk az eredeti eszközön.

Mindezek alapján látható, hogy a GATT funkciók továbbítása a megoldás jellegéből fakadóan viszonylag egyszerűen megoldható az initiator szempontjából transzparens módon.

2.3.5. Lekapcsolódás a mediátor eszközökről

A hálózat szétesését az initiator eszköz lekapcsolódása idézi elő, ekkor ugyanis a mediátor eszköz észlelve a Centrallal alkotott kapcsolat bontását, jelen megoldás során bontja a kapcsolatát minden egyes Peripheral eszközzel (és így tovább a mediációs hirerarchia legalsóbb szintjéig). Ebből fakadóan a hálózat viszonylag gyorsan szétesik. A kapcsolat szétesése pedig jelen állapot szerint (a QoS paraméter nem definiált voltából fakadóan) egyben a közvetített szolgáltatások törlését is magában hordozza.

2.4. A megoldás előnyei és skálázhatósága

A megközelítés legfontosabb előnye, hogy jellegéből fakadóan egyáltalán nincs szükség a hálózat ismeretére a többes ugrásos kommunikáció megvalósításához. Ez mind az alkalmazások fejlesztését, mind a management funkciókat jelentős mértékben egyszerűsíti, hiszen nincs szükség adott esetben semmilyen adatbázisra egy adott célra fejlesztett modulokat és szolgáltatásokat felölelő rendszerben, elegendő csak a definiált

szolgáltatás és a kapcsolódó Characteristic és deszkriptor UUID-k ismerete (ami önmagában meghatározza az adott szolgáltatást), valamint az SMS jelenléte a működés megvalósításához.

További fontos előnye a megoldásnak, hogy lehetővé teszi a heterogén Bluetooth LE rendszerek (modulok és szolgáltatások) együttműködését is, hiszen lényegében nem vár el semmilyen különleges metódust a végpontoktól (részben kivételt képez ez alól az Initiator, ám ez jellegéből fakadóan intelligensebb készülék), csak a GATT által meghatározott procedúrákat (implicit, vagy explicit módon) játssza át, valamint hajtja végre adott esetben. Ennél fogva teljesen (és visszafelé) kompatibilis bármilyen Bluetooth LE eszközzel, s így nincs szükség az adott esetben harmadik fél által gyártott modulok semmilyen módosítására, amennyiben SMS-t implementáló modulok is jelen vannak.

Jelentős előny ezen felül, hogy a többes ugrásos kommunikáció és felderítés végrehajtásához csak a piconet csatornákat használja, így a hirdetményekbe szabadon beágyazhatók a különböző gyártó- és modulspecifikus információk, ennél fogva elméletben tökéletesen megfér bármilyen céleszközben, bármilyen szolgáltatás mellett, azaz általánosnak tekinthető.

Bár alapvetően nem volt cél, a megoldás bizonyos értelemben tekinthető az NDN részleges implementációjának Bluetooth LE hálózatokban (ami szintén előny, hiszen alapvetően egy Future Internet architektúrába való integrációt segít elő). Ugyanakkor megjegyzendő, hogy a tényleges megvalósításhoz az Initiator eszközben ehhez definiálni kell egy alkalmas interfészt, mely a kiolvasott adatokat a felsőbb rétegek számára elérhetővé teszi (ekkor nyilván csak olvasható adatokat deríthetünk fel, valamint csak egyértelmű neveket használhatunk, ami visszalépés az eredeti megoldáshoz képest, hiszen a végpontok ismeretét igényli). Ezen felül a biztonság és a QoS kérdéskörét is módszeresen körül kell járni, és a megfelelő funkciókat át kell ültetni ezen szolgáltatásba.

A skálázódás tekintetében a megoldás előnyei közé tartozik, hogy bármilyen bonyolultságú és összetételű hálózatban képes a szolgáltatások közvetítésére, hiszen a mediációs hierarchia bármelyik szintjén elhelyezkedhetnek a mediátor egységek. Az egyetlen szükséges feltétel, hogy minden eszköznek (meditátor és nem mediátor) legyen rálátása legalább egy mediátor egységre.

Ugyanakkor szintén a skálázódás tekintetében bizonyos szempontból hátrány a megoldás memóriaigénye, hiszen míg önmagukban az egyes GATT struktúrák nem foglalnak sok helyet, addig az aggregáció folyamán a mediátorokban ezek együttesen

vannak jelen, ami bizonyos eszközszám és mediátor hierarchia szint felett kritikus méreteket ölthet. Az egyes bejegyzések számának elméleti maximuma a Bluetooth specifikációjában meghatározott handle értékek maximumával arányos, ami 65535 (16 bites előjel nélküli integer).

Bár ez alapvetően soknak tűnhet, vegyük figyelembe, hogy egyetlen szolgáltatás, három definiált Characteristic (deklarációs és érték handle-ök) és az azokhoz egyenként tartozó három deszkriptor (pl. Client Char. Conf. Descriptor, User Description Descriptor, NDN Descriptor) már önmagában 16 handle-t ölel fel, ha a hálózatunkban 100 db ilyen eszköz helyezkedik el, akkor az összesen 1501 (1600-99, hiszen a szolgáltatásokat csak egyszer deklaráljuk) bejegyzésnek felel meg az initiatorhoz legközelebbi eszközön (amennyiben csak azon keresztül érhető el a hálózat), ami önmagában még mindig nem sok (az elméleti maximumhoz képest).

Azonban ezen bejegyzések változó hosszúságúak (a deklarációk jellemzően 5–10 byte hosszúak, 128 bites UUID-k esetén kb. 19–24 byte), emellett az értékekhez külön memóriaterület tartozik (ami jellegéből fakadóan változó hosszúságú lehet a típustól függően, jelen számításban vegyük az Attribute Protocolban definiált minimális MTU-t amely 23 byte-nak felel meg [17]). A Service kivételével minden egyes alsóbb szintű deklarációhoz érték is tartozik, ami eszközönként összességében közel 300 byte, így egy kritikus ponton 100 eszköz esetén 30 KB-nyi memóriára is szükség lehet, ami beágyazott rendszerek (8–16 KB RAM + 128–256KB Flash) esetén - még, ha a flash memóriában is kerül tárolásra, akkor is - igen nagy kihívás, a jellemzően ~100KB nagyságú BLE stack-ek mellett (mindemellett a flash memória sokkal kevesebb írási ciklust visel el, mint a RAM).

További fontos szempont az egyes eszközök fogyasztása, mely szintén a mediátor eszközök esetén lehet kritikus kérdés. A gyors felderítések érdekében érdemes a throughputot maximalizálni, ami a „connection event”-ek minimális hosszúságúra állításával (7,5 ms) tehető meg. Ekkor a rádió közel 10%-os kitöltési tényezővel dolgozik, ami a jellemző ~15 mA-es áramfelvétel [36] mellett 1,5 mAh fogyasztást idéz elő, így a tipikusan 4–500 mAh-s gombelemek (pl. CR2450) igen gyorsan (kb. 10–15 nap alatt) lemerülnek. Bár a kitöltési tényezőt nagyságrendekkel is csökkenteni lehet, az összességében a rendszer késleltetését is növelni fogja. Ez a jelenség egyfajta „trade-off”-nak tekinthető, mely az adott alkalmazási terület kérdéskörébe tartozik.

Bár ezek alapvetően csak becslések, a valóságtól egyáltalán nem elrugaszkodottak. Így ezen kisebb számításokból kiindulva olyan eszközökön, melyeket kis kapacitású akkumulátorok üzemeltetnek, valamint mikrokontrollereken futnak a

különböző szolgáltatások, igencsak erős megkötésekkel (közvetíthető szolgáltatások számának korlátozásával, valamint a késleltetés növelésével) alkalmazhatók általánosan mediátor egységeknek.

Ugyanakkor akadnak olyan készülékek, melyek tápellátása a hálózatról történik (okos hálózati aljzatok, villanykörték, kapcsolók), így ezen eszközök esetén nem olyan erős szempont a fogyasztás, hogy ne lehetne a késleltetést minimalizálni. Ezekben az esetekben csak a közvetítendő szolgáltatások számának maximuma az, ami korlátként megmarad. Ezen utolsó korlát csak olyan eszközök esetén engedhető el, amelyek alapvetően nagyobb intelligenciával vannak felvértezve, így nagyobb a rendelkezésre álló memória is (pl. „okos" televíziók, telefonok, és egyéb készülékek).

Látható tehát, hogy bár a megoldás elméletben általánosan alkalmazható, a gyakorlatban ennek tényleges megvalósítása korántsem triviális kérdés a szükséges memória és a teljesítmény skálázódását figyelembe véve.

2.5. A továbbfejlesztés irányvonalai

Bár a szolgáltatás számos sarkalatos pontja ismertetésre került, mégis maradtak olyan funkciók, melyek nem kerültek kidolgozásra. Az egyik legfőbb ilyen pont a biztonság kérdésköre, mely alapvető feladat a kiterjedtebb ipari megoldásoknál. Az NDN architektúra ezt az adatok szintjén definiálja, ami részben ugyan át is emelhető, de sajnálatos módon nem teljesen, hiszen míg ott a hálózat egységein akármilyen módosítást megengedhetünk, addig ezen hálózatokban, ha meg szeretnénk tartani a visszafelé való kompatibilitást, akkor csak a specifikált Bluetooth Low Energy autentikációs és titkosítási eljárásokat alkalmazhatjuk, melyek összeférhetőségének és „átfordíthatóságának" vizsgálata az NDN-ben definiáltakkal korántsem triviális feladat.

Az eddig sokat emlegetett QoS paraméter szintén kidolgozatlan pont. A rendszer skálázhatóságáról leírtak fényében egyáltalán nem elhanyagolandó szempont, hiszen a különböző tárolási- és energiakapacitással rendelkező eszközök különböző korlátokkal rendelkeznek, így olyan eljárásokat szükséges kidolgozni, amelyek a különböző szkenáriókban is biztosítják lehetőség szerint az összes szolgáltatás közvetíthetőségét anélkül, hogy az akkumulátoros egységek fogyasztását lényegesen megnövelnénk, valamint a flash memóriát „idő előtt" elhasználnánk.

Az ilyen eszközökben általában deklarálásra kerül egy „Battery Service", mely az akkumulátor állapotát hívatott mutatni egy definiált Characteristic értékén keresztül. Emellett a „Device Information Service", valamint a kötelezően deklarálandó „Generic Access Profile" szolgáltatás is árulkodó lehet, melyhez a felderítések során

hozzájuthatunk, így a mediáció során priorizálhatjuk az egyes mediátor eszközöket [37]. Mindemellett eldönthetjük azt is, hogy két irányban is frissüljön-e az aktuális nézet, aminek következtében minden kitüntetett mediátor eszközön egy és ugyanaz a kép alakul ki az adott szolgáltatások hálózatban való jelenlétéről és helyéről, így bármely mediátor eszközön keresztül újabb felderítés nélkül is hozzáférhetünk bármely más eszköz GATT struktúráihoz.

Amennyiben a bejegyzések frissítési periodicitását és élettartamát is figyelembe vesszük a QoS paraméter meghatározásánál, úgy különböző dinamikájú (statikus, kvázi statikus, avagy dinamikusan változó topológia), és heterogén kapacitású Bluetooth LE hálózatokban is képesek lehetünk megfelelően konzisztens nézetet kialakítani bármely mediátor eszközön. Mindez összességében egy olyan optimalizálási feladat, amelyet mind az initiator, mind a mediátorok maguk (minthogy semmi nem akadályozza azt, hogy egyedi nézeteket alakítsanak ki és a kiolvasott értékek alapján döntsenek) is el tudnak végezni a megfelelő információk összegyűjtésével.

A Timeout értékek és a szövevényes topológia támogatása szintén kidolgozandó és erősen összefüggő pontok, hiszen a bejövő Interestek függvényében kell a mediátoroknak dönteni arról, hogy mekkora késést engedhet meg a következő mediátornak, az eredeti Interest forrása által meghatározott időintervallumon belül. Mindez azért sem triviális feladat, mert a megoldás jellegéből fakadóan nem tudhatjuk, hogy a mediációs hierarchia melyik szintjén foglalunk helyet, valamint, hogy hány ugrásra van tőlünk a legtávolabbi eszköz (pl.: alkalmazhatunk próbákat, valamint megjegyezhetjük, hogy melyik eszközt milyen Timeout értékkel érhetünk el, a szélességi, vagy mélységi keresés alkalmazásának kérdése ekkor válik kritikussá). A legegyszerűbb megoldás nyilván a hálózat topológiájának ismerete lenne, hiszen ekkor tudnánk előzetes becslésekkel élni, azonban ez összességében további memóriaigényt generálna.

Mindez összességében igen jól mutatja, hogy a megoldás korántsem tekinthető teljesnek és számos olyan pont vár kidolgozásra, melynek segítségével a működés az igényeknek megfelelően finomhangolható.

IV. Kísérleti implementáció

Jelen fejezetben bemutatom az implementációs környezetet, így a használt eszközöket és szoftvermodulokat. Ezt követően röviden összefoglalom a megvalósítás során tapasztalt különböző anomáliákat és problémákat. Bemutatásra kerül azon kísérleti elrendezés, melyen a verifikációt a különböző a funkciók tesztelésével végeztem. Mindezeket követően röviden értekezem a tényleges implementáció optimalizálási lehetőségeiről.

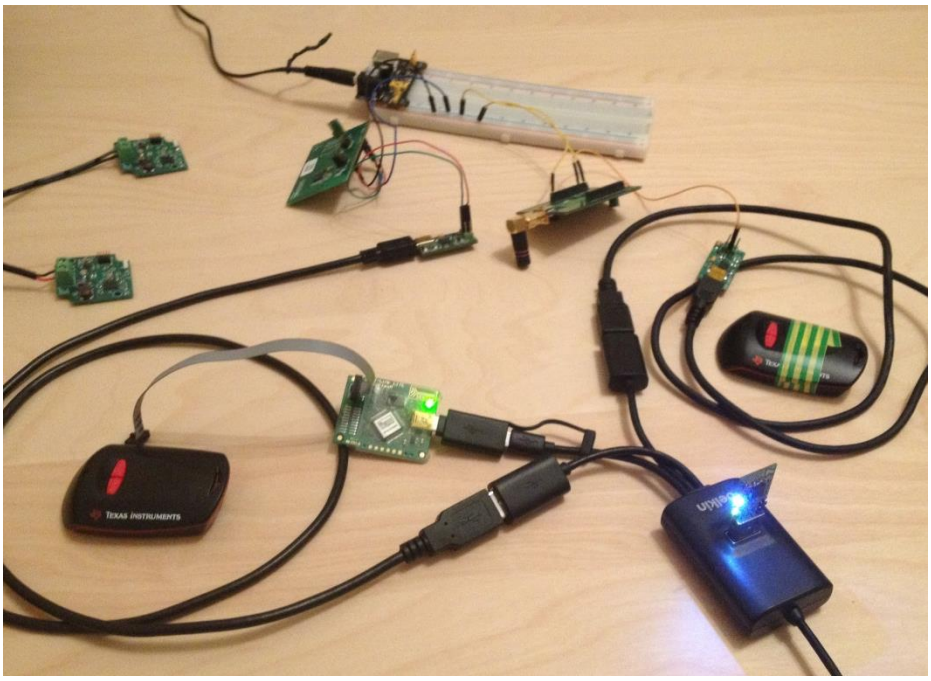
1. Implementációs környezet

A megvalósítás során három gyártó eszközkészletét alkalmaztam. Az egyik a Texas Instruments által gyártott CC2540 Mini Development Kit [38], melynek a „kulcstartó” eszközeit a hozzáadott demo alkalmazással programoztam fel (ebben számos szolgáltatás deklaráltak, melyek különböző funkciókat valósítanak meg).

A második gyártó a Nordic Semiconductors, melynek az nRF51822 Development kitjét alkalmaztam [39]. Ennek részét képezi két modul, melyekhez a jelenleg alfa fázisban levő S130 SoftDevice [40] BLE stacket, valamint hozzáadott példakódját (a mediátor eszközök működésének megfelelően átírva) alkalmaztam. A fejlesztést Windows 7 (x64) rendszerben a Keil µVision (v5.11.1.0) IDE (Integrated Development Environment) próbaverziójának (32KB maximális kódméret) segítségével végeztem. Ezen felül a készlet USB modulján keresztül a hozzáadott MasterControlPanel program segítségével az Initiator elvárt folyamatokat manuálisan végeztem el.

A harmadik gyártó a BME-Infokom Innovátor Nonprofit kft., melynek prototípus Bluetooth Low Energy modulját egy olyan próbaalkalmazással vérteztem fel, mely egy szívritmusmérő eszközt imitál. A Texas Instruments és ez utóbbi modulok ennek megfelelően azon eszközök szerepét töltötték be, melyeken a különböző szolgáltatások felderítendők, valamint a mediátor egységeken keresztül közvetítendők.

A mediátor eszközökön a működést egy soros interfészen keresztül kiadott üzenetek által, USB-UART átalakítók segítségével, valamint a RealTerm [40] program felhasználásával monitoroztam. A mérési elrendezést a következő ábra mutatja.



27. ábra – Mérési elrendezés

2. A fejlesztés során tapasztaltak

Az SMS implementációja során az első probléma, melybe belefutottam, a korábban részletesen taglalt memóriakapacitás kicsinysége, mely a szolgáltatások felderítésekor kapott adatok ideiglenes megjegyzésére (és a forwarding funkciókhoz szükséges lokális és külső handle párok perzisztens tárolására) létrehozott struktúratömbök definiálásakor előjött (a linker jelezte, hogy túlfutottam a korláton). Ennek megfelelően erős optimalizálásra kényszerültem. Megjegyzendő, hogy az egyszerűbb fejlesztés és hibakeresés érdekében csak és kizárólag statikus memóriakezelést végeztem, amivel egyértelművé vált a kód összeállításánál, hogy a hiba a memóriahasználatban van-e (dinamikus memóriakezeléssel sokkal rugalmasabban oldható meg a feladat). Első ötletemben egy olyan megközelítést alkalmaztam, melyben a felderítés során csak egyetlen struktúrát alkalmazok az adott szolgáltatás felderítésére, majd a következő felderítése előtt regisztrálom az adott közvetítendő szolgáltatást.

Ekkor azonban szembesültem azon problémával, hogy egy included service hozzáadása csak akkor (és csak a fő szolgáltatás regisztrációjakor) tehető meg, ha korábban már regisztráltam a másodlagos (included) szolgáltatást. Bár ez önmagában nem jelente problémát (hiszen töröljük, regisztráljuk a közvetítendőt, majd újra regisztráljuk a mediációs szolgáltatást), az alfa fázis folyományaként az API-ból

kimaradt azon funkció, mellyel futási időben törölhető lenne egy korábban regisztrált szolgáltatás. Ez alapjaiban lehetetlenítette el a koncepció korábban leírt módon való megvalósítását.

A probléma megoldásaként az SMS-en kisebb módosítást (inkább „hack-elést") végeztem, éspedig bevezettem egy új Characteristic-et „Mediation Offset" néven, melyet hozzáadva az adott szolgáltatás UUID-jához regisztrálhatjuk elsődleges (primary) szolgáltatásként anélkül, hogy ütközés lenne az UUID-k között. Ennek megfelelően a regisztráció additív jelleggel megvalósítható bármely szolgáltatás esetén.

A törlés azonban továbbra sem lehetséges, így a lekapcsolódáskor nem törlődnek a közvetített GATT struktúrák, ami összességében jelen feladatban inkább előny, hiszen tesztelhető a megoldás olyan esetekben, amikor elvész a kapcsolat, azonban topológia nem változik és a GATT struktúrák megmaradnak. Megjegyzendő, hogy bár alapvetően így is működhetne a szolgáltatás, a QoS paraméter kidolgozatlanságából fakadóan kezelhetetlen a bejegyzések élettartama („hasraütés-szerűen" pedig továbbra sem tartottam helyesnek a működés definiálását, ezért döntöttem a törlés mellett, hiszen ez tekinthető a legegyszerűbb, alapértelemezett esetnek). A „Mediation Offset" használatát természetesen átvezettem a felderítésekbe is.

Ugyanakkor az eredeti probléma továbbra sem szűnt meg, hiszen semmi nem garantálja, hogy a következő eszközön nem lesz egy újabb, ezzel azonos szolgáltatás, ha pedig már regisztráltunk egy új szolgáltatást egy olyan után a stackben, amelyhez találtunk egy újabb Characteristic értéket, úgy azt már nem adhatjuk hozzá. Ennek megfelelően a teljes felderített struktúratömböt fenn kell tartani a mediációs folyamat befejeztéig. Ezen apró hiányosság alapjaiban korlátozta a megvalósítás során alkalmazható maximális közvetíthető szolgáltatások számát.

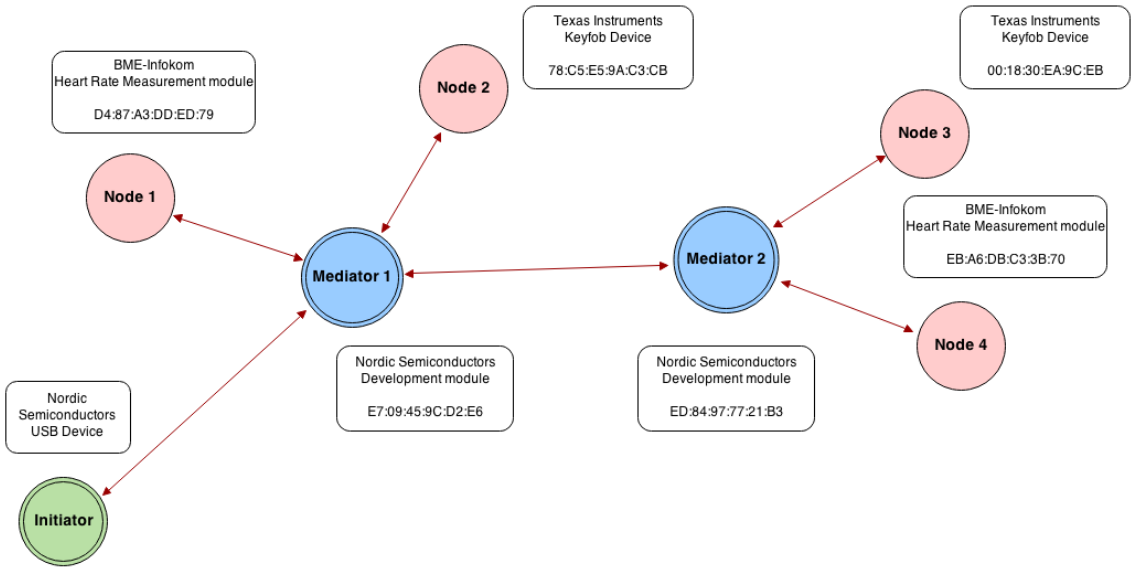
További nehézséget okozott, hogy a felderítést és regisztrációt követően sem volt módom a struktúratömbtől megszabadulni. Ennek oka, hogy a lokális GATT adatbázison végezhető operációk közel sem szimmetrikusak a Bluetooth LE és az API oldalakat tekintve. Míg a rádiós oldalról van mód az UUID alapú keresésre (mely egy lokális handle értéket ad vissza), addig erre az API-ból nincs lehetőség.

Mindennek megfelelően sajnálatos módon, közel a teljes deklarációs struktúrát duplikált módon fenn kellett tartanom (ugyanakkor legalább a tényleges értékeket nem) a működés ideje alatt. Ami összességében erős korlátokkal rendelkezett (max. 6 db service, azokon belül max. 4db Characteristic, azokon belül max. 4db descriptor – ahogyan az a Függelék 2.1-es pontjában látható).

Egyébiránt a fejlesztés az eszközökön viszonylag egyszerű volt, bár megjegyzendő, hogy a próbaalkalmazás eseménykezelő ciklusára építettem az alkalmazást, amit több esetben is rekurzívan szükséges hívni (egy várt esemény érdekében), ebből fakadóan bizonyos esetekben meglehetősen lelassul az események feldolgozásának folyamata. A teljes átstrukturálás és egy új eseménykezelő rutin megírása az időbeli kapacitásaimat meghaladta, így eltekintettem tőle.

3. Verifikáció

A verifikáció során az egyes eszközök által felderített eszközök listáján mesterséges szűrést alkalmaztam a megfelelő láthatósági gráf kialakításának érdekében, az initiator eszközön pedig célirányosan egy mediátor eszközhöz csatlakoztam, s azon végeztem el a szükséges feladatokat a mediációs folyamat elindításához. Az így kialakított láthatósági a gráfot a következő ábra mutatja.

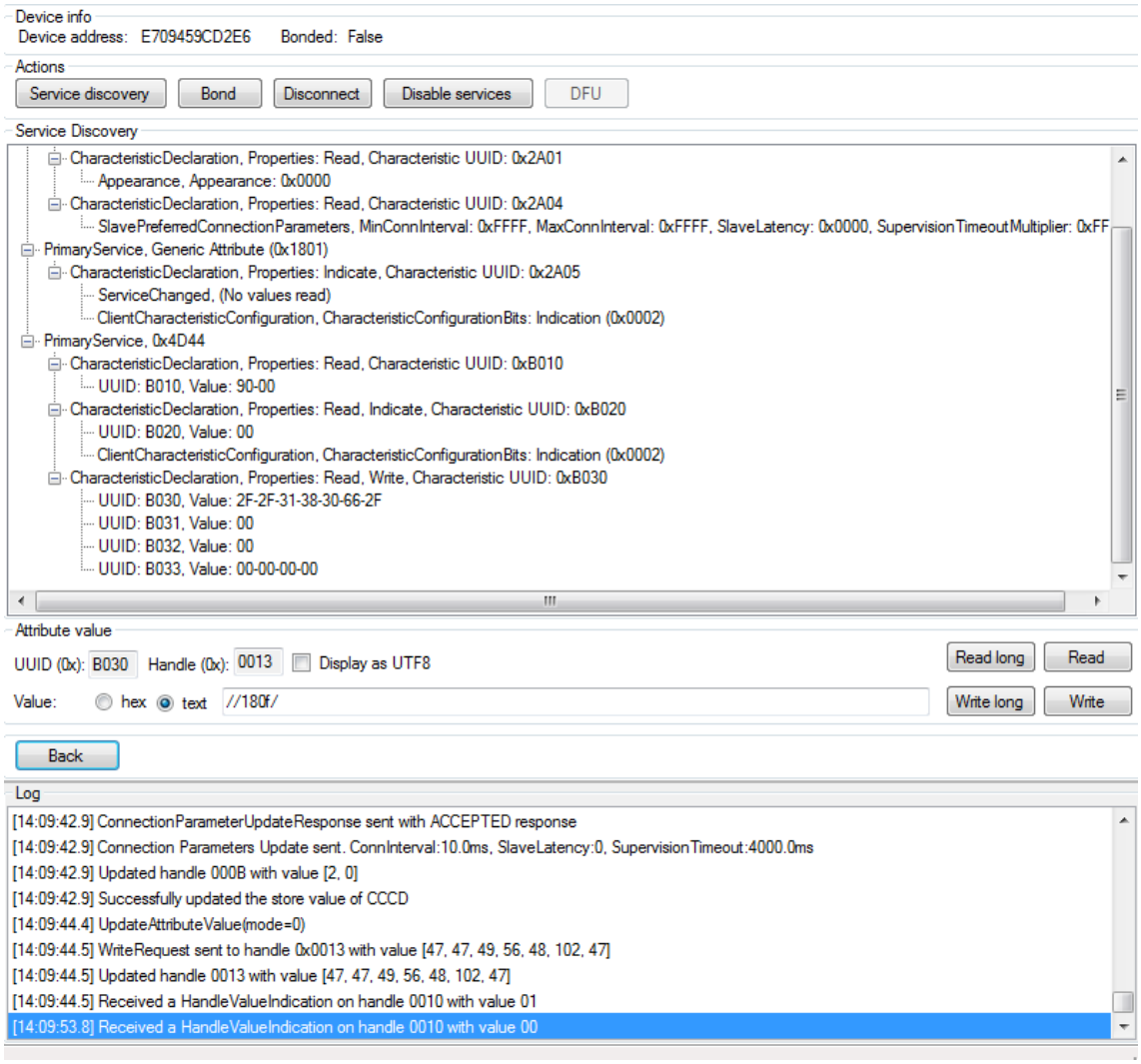


28. ábra – A mesterségesen kialakított verifikációs topológia

Megjegyzendő, hogy az eszközök felderítését a mediátor modulok (jelen verifikációs rendszerben) inaktív állapotban is periodikusan végzik, így egy adott eszköz jelenlétének vizsgálata egy adott mediátor hatósugarában a beérkező Interestet feldolgozását követően egyből megtörténhet, ami összességében gyorsítja a mediációs folyamatot (tekinthető lehet a QoS által beállítható optimalizációs lehetőségnek is).

3.1. A felderítés és olvasás tesztelése

A felderítés során a korábban is említett „Battery Service”-t kereste a rendszer. Az ehhez szükséges név: „**//180F/**” (0x180F a definiált UUID) [37].



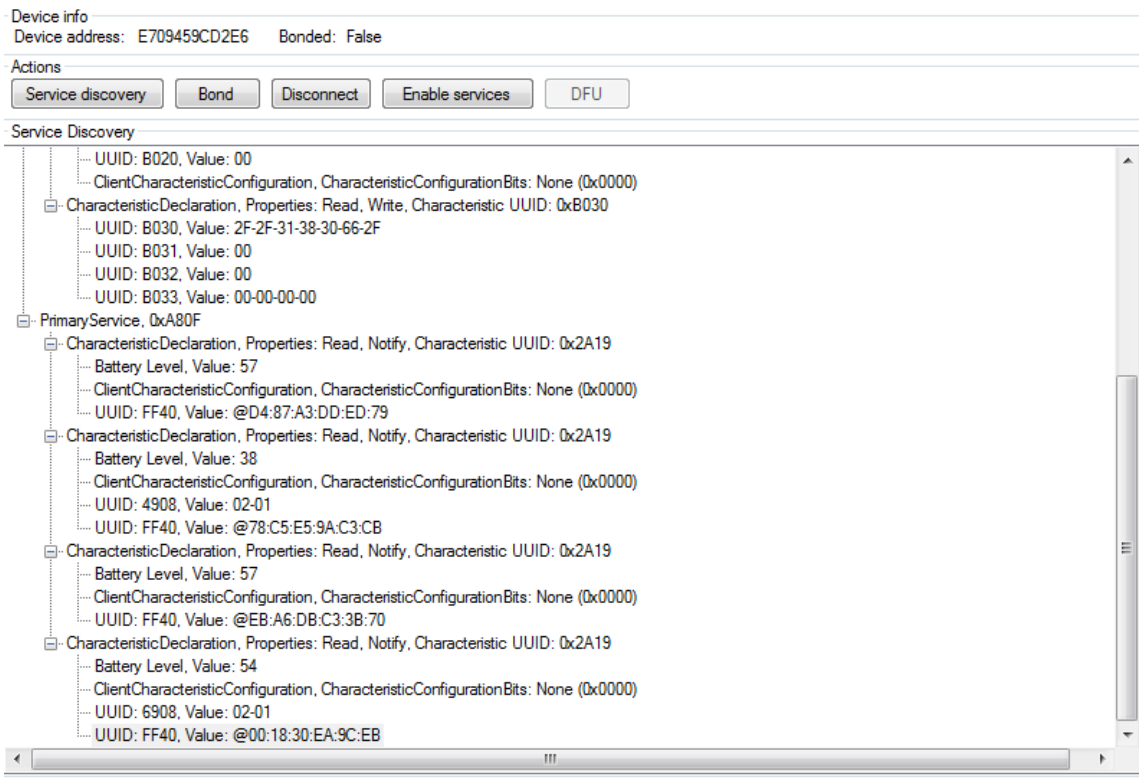
29. ábra – A felderítés során kapott jelzések

A MasterControlPanel képernyőképén jól kivehető az SMS GATT struktúrája (Service UUID: 0x4D44). Az első definiált Characteristic a korábban említett „Mediation Offset” (UUID: 0xb010), melynek értéke 0x9000. Azt követi a „Mediation Signal Characteristic” (UUID: 0xb020) (melyhez egy, az Indication funkcióra való „feliratkozást” elősegítő Client Char. Conf. Descriptor került deklarálásra). Utolsó a sorban az „Interest Characteristic”(UUID: 0xb030), melyhez a szükséges deszkriptorok is deklarálásra kerültek (rendre: QoS, Timeout és Nonce).

Láthatóan a folyamat kezdeményezése a [14:09:44.5]-ös időpillanatban ment végbe (óra:perc:mp.tmp formátumban), ahol is beírásra került az összeállított név az

Interest Characteristic értékmezejébe. Ugyanezen tizedmásodpercen belül történt meg a karakterlánc tokenizálása és a „MEDIATION_IN_PROGRESS” (0x01) jelzésüzenet küldése egy „Handle Value Indication” (UUID) formájában. Mintegy 9,3 másodperccel később érkezett meg a „MEDIATION_READY” jelzésüzenet (0x00), azaz ennyi időbe telt jelen hálózatban egy adott szolgáltatás felderítése (a releváns eseménynapló részlet a Függelék 3.1. pontjában található meg).

A szükséges idő hosszúsága, egyrészről a sikertelen kapcsolódási kísérletekből fakad (egyes eszközökre csak harmadjára sikerült kapcsolódni, ez a folyamat hosszabb szkennelési ablakokkal biztosabbá tehető), másrészről a technológia azon funkciójával függ össze, melynek megfelelően a Peripheral eszközök módosíthatják a connInterval értékét, az alacsonyabb kitöltési tényezővel történő rádióhasználat, s így az alacsonyabb fogyasztás érdekében (vagy bármilyen más okból). Tekintettel arra, hogy e felett nem feltétlenül rendelkezhetünk (tipikusan egy „multi-vendor” konfigurációban) a különböző felderítések ideje akár nagyságrendekkel is eltérhet.

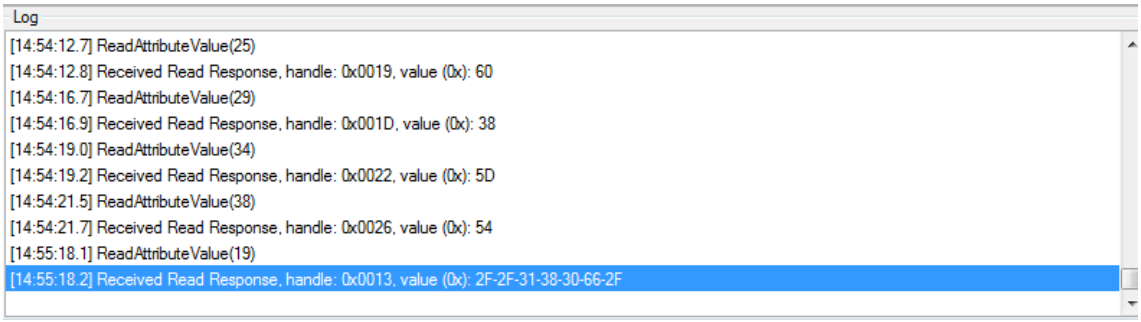


30. ábra – Az újonnan vérehajtott GATT felderítés eredménye

A felderítés eredményeként előállt az initiatorhoz legközelebbi mediátor egységen a kívánt és minden szempontból konzisztens (hiszen a lekérdezés során ki is olvassuk a forrásból, ami olvasható) nézet. Az NDN deszkriptorok (UUID: 0xFF40) jól mutatják, hogy melyik eszközre vonatkoznak az egyes értékek, valamint jól látható a

mediációs ofszet addíciója is. Megjegyzendő, hogy az újonnan kiadott GATT felderítésre nem minden esetben érkezett megfelelő válasz, így több esetben le kellett kapcsolódni, majd újrakapcsolódni a mediátor egységre. A jelenség (minthogy tisztán GATT specifikus művelet, melyet az alkalmazott BLE stack szabályoz, így értesítést nem is kapunk róla a beágyazott alkalmazásban) feltehetően az alfa fázis aktuális hiányosságaiból, vagy a helytelen eseménykezeléséből fakad.

Bár ezen folyamat egyben az olvasások működését is verifikálta, a késleltetésre vonatkozóan nem nyújt megfelelő információt, így az egyes értékeket a GATT felderítést követően újra megtettem egyenként.



31. ábra – A közvetített GATT olvasások késleltetése

Viszonyítási alapként végrehajtottam egy, a legközelebbi mediátor egységen levő Characteristic érték olvasását is, mely így a Windows alkalmazás (MasterControlPanel) által okozott késleltetést is jól mutatja, hiszen ezen a kapcsolaton nem módosult a 10 ms-os connInterval érték, így az olvasás itt mindenképpen gyorsabban megtörténik. A belső eseménynaplója által rögzített események ez esetben:

...14:55:18.1324...Serial port write: ...00-07-00-03-00-04-00-0A-13-00

...

...14:55:18.1764...Serial port read: ...20-0C-00-08-00-04-00-0B-2F-2F-31-38-30-66-2F

Amiből igen jól látszik, hogy az alkalmazás kerekítése tekintélyes pontatlanságot okoz, hiszen a közel 44 ms-os késleltetés helyett 100 ms-ot mutat. Ez a felderítés során marginális különbséget jelentett, ám ez esetben nem mindegy.

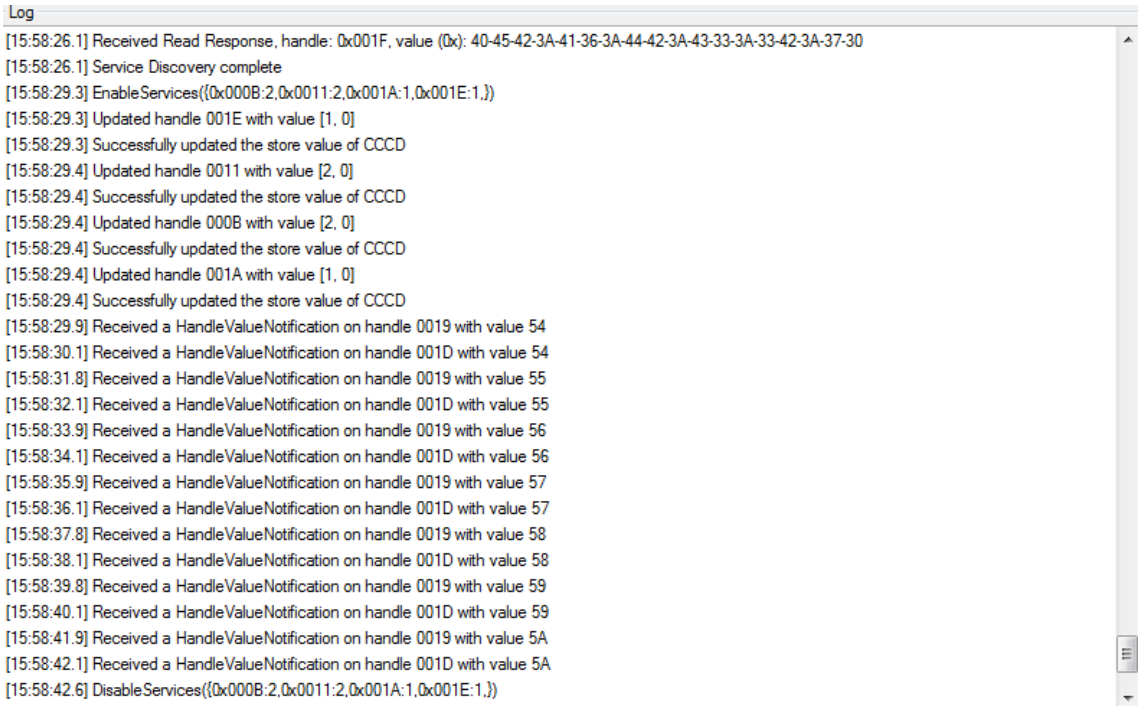
Az olvasások a felderítés eredményét mutató ábrán látható sorrendben történtek. Látható, hogy míg az egyik két ugrásra levő modulon (BME-Infokom) a kiolvasás a belső eseménynapló alapján kb. 100 ms alatt megtörtént, addig a másikon (Texas Instruments) ehhez 120 ms-ra volt szükség (a connInterval értéke különböző a két modul esetében). A három ugrásra levő modulokon a BME-Infokom eszköze esetén 150

ms-ra volt szükség, míg a Texas Instruments moduljáról a kiolvasás 190 ms-ot igényelt (Függelék 3.2. pont).

Bár összességében egyáltalán nem tekinthető rossz eredménynek, fontos kiemelni, hogy ezen mérést a loginformációk konzolra való kiíratása (melyet számos esetben megteszek) is torzítja (115,2 Kbaud esetén egy 100 karakter hosszú üzenet kiírása közel 1 ms-ot igényel).

3.2. Az írás és a notifikációk továbbításának tesztelése

A notifikációk aktiválása az egyes Characteristic értékekhez tartozó Client Char. Conf. Descriptor 0x0001 értékre történő módosításával történhet meg. Így ezen módszerrel egyszerre két funkció tesztelését hajtottam végre. Az írás érvényrejutását ennek megfelelően az első notifikáció megérkezése jelentette, ami egyben azok továbbításának működését is verifikálta.



32. ábra – Az írás és a notifikációk továbbítása

Az egy „mozdulattal” történő összes notifikáció aktiválásához a MasterControlPanel „Enable Services” funkcióját használtam. Ennek megfelelően látható, hogy az írás a 0x001E (három ugrásra levő modul) és 0x001A (két ugrásra levő modul) handle értékeken végrehajtódott. Jelen esetben csak a BME-Infokom moduljait alkalmaztam, ugyanis Texas Instruments eszközökön, bár a notifikáció beállítására lehetőség volt, azok nem küldtek egyet sem belátható időn belül (direkt kapcsolat esetén

sem). (Tekintettel arra, hogy jelen esetben csak két eszközön hajtottunk végre GATT felderítést, jelen esetben közel 4 másodperc is elegendőnek bizonyult erre a feladatra.)

Az írás érvényre jutásának késleltetése is igen jól látható, hiszen a két első notifikáció megérkezése között 200 ms telt el, ami a továbbítás során nem is nagyon ingadozott a későbbiekben (a Függelék 3.3-as pontjában jól látható).

A notifikációk küldésének periodicitása 2 mp volt, amit láthatóan tartott ezen idő során a mediátor eszköz, azonban megjegyzendő, hogy hosszabb távon (2–3 óra) akadtak anomáliák, melyek során megtelt a BLE stack notifikációs puffere, s így az nem továbbított több értesítést. Mindemellett a nagyobb gyakorisággal (20 db/mp) küldött notifikációk sem gyakoroltak jótékony hatást a működésre, hiszen ez esetben pillanatok alatt megtelt ugyanezen tároló a mediátorokon. Mindez jelen esetben betudható mind az általam írt eseménykezelő ciklusnak rekurzív hívhatóságának, mind az alkalmazott BLE stack kezdetlegességének.

Ugyanakkor jól látható, hogy megoldás képes teljesíteni a tőle elvártakat, az alkalmazott BLE stack korlátainak figyelembe vételével. Tekintettel arra, hogy az indication funkció tulajdonképpen csak egy megbízható notifikáció (mely jellegéből fakadóan kisebb, vagy egyenlő gyakoriságú indikációs események továbbítására alkalmazható), annak tesztelésétől jelen esetben eltekintettem.

4. Az implementáció továbbfejlesztéséről

Amint az korábban is említésre került, az eseménykezelés felépítése korántsem triviális feladat, így a jövőben egy olyan ciklus megírása a cél, melyben az SMS állapotgép jellegű megvalósítása fog szerepelni, elkerülendő a felesleges rekurzív hívásokat, melyek a fő végrehajtási ciklusba való visszajutást megnehezítik. Mindez összességében bár növelni fogja a kódméretet (mely jelen esetben ~32 KB), a működést robosztusabbá teszi.

A korlátozott memóriaméret szintén probléma, így megfontolandó a flash memória fennmaradó részének használata (ügyelve az egyenletesen elosztott blokkhasználatra, valamint a hibás blokkok elkerülésére, biztosítandó a minél hosszabb üzemidőt).

A megfelelő GATT funkciók hiányában ideiglenesen megfontolandó a GATT struktúrák offline (amikor nincs élő BLE kapcsolat) kezelésekor a BLE stack újraindításával törölni a regisztrált struktúrák összességét, majd visszaépíteni az

aktuálisakat (ez egyben a puffereket is újrainicializálja, így a kezdetleges stack is hosszabb távon alkalmazhatóvá válik).

Azonban amíg a stack nem képes a GATT struktúrák futási időben való törlésére, addig a közvetített szolgáltatások SMS-be való ágyazása másodlagos szolgáltatások formájában, a jelen megkötések szerint kivitelezhetetlen. Ugyanakkor a módosítás (mediációs ofszet) jellegéből fakadóan működőképessé teszi azt ily módon is. Így az SMS-t érintő további fejlesztések és tesztelések „zavartalanul” folytatódhatnak, a BLE stack korlátainak figyelembe vételével.

V. Továbblépési lehetőségek

Jelen fejezetben olyan lehetőségeket taglalok, melyek a megoldás szellemiségéből fakadóan nem feltétlenül csak a Bluetooth LE hálózatokban, hanem bármilyen megfelelően definiált alkalmazási keretrendszerrel (objektumok, paraméterek, üzenetek, struktúrák, szerepek, stb. – pl. BACnet [42], KNX [43], LonWorks [44], stb.) konform ad-hoc, avagy infrastruktúra-alapú, vezetékes avagy vezeték nélküli hálózatban – az SMS NDN alapú koncepciójának átültetésével – alkalmazhatóak lehetnek adott esetben.

1. Adatelérés reguláris kifejezésekkel és feltételvizsgálatokkal

Az első ilyen lehetőség az adatelérések, s így a nevek kiegészítése egyszerűbb reguláris kifejezésekkel. Bár az NDN alapvetően kiköti a globálisan elérhető adatok neveinek egyértelműségét, ez az Interest-ekben korántsem feltétlenül követendő. Az NDN architektúrában az Interest csomagok rendelkeznek különböző finomhangoló mezőkkel (úm. childselector, minsuffix, maxsuffix, stb.) [31], melyekkel névhierarchia alsóbb részein végezhetünk szűrést, ugyanakkor nincs mód a felsőbb elemek hangolására, így minden esetben ismernünk szükséges az adatok nevének megfelelő előtagjait.

Azonban egy jól definiált (és legfőképpen véges) névtérben a többértelmű, és ennél fogva több választ is okozó Interest-ek összeállítása nem eredményez végtelen válaszlehetőséget, így túlontúl nagy forgalmat sem. Ezt példázta a verifikáció során is alkalmazott „**//180F/**” lekérdezés is, hiszen a mezők üresen hagyásával gyakorlatilag az összes lehetséges választ kijelöltük. Ugyanakkor alkalmazhatnánk a következő karakterláncot is: „**/(a | b | c)/(180F & 180A)/**”, mely egy olyan elosztott nézetet alakít ki a legközelebbi mediátor egységeken, amelyben csak az „a”, „b”, és „c” nevű eszközök szerepelhetnek és azok is csak akkor, ha mindkét kijelölt szolgáltatást tartalmazzák.

Ezen gondolatmenetet továbbfűzve vezessük be a feltételvizsgálatok lehetőségét a Characteristic értékek szintjén. A sztenderd C szintaktikát követve a „**///(2202 >= 2)**” név-prefixum csak olyan GATT struktúrákat fog megjeleníteni a nézetben, melyekben a „**2202**” nevű Characteristic nagyobb, vagy egyenlő kettőnél. Az érték vizsgálata minden további nélkül megtehető, hiszen a mediáció során (tudjuk, hogy olvasható, vagy azért mert egy saját szolgáltatás része, vagy azért mert a szabványos) ki is olvassuk.

Az ilyen és ehhez hasonló vizsgálatok segítségével nem csak a mediációs folyamatokhoz szükséges memóriakapacitást csökkenthetjük, de a felderítéseket is következetesen végezhetjük, „brute-force” jellegű teljes keresések helyett, melyeket ráadásul kitüntetett szolgáltatások esetén elegendő csak egyszer végrehajtani.

Amennyiben egy felsőbb menedzsment rétegből végzünk, adott átjárókon keresztül akár több szájton keresést (ekkor Bluetooth LE, valamint NDN esetén a névtér „/site/location/service/characteristic” formátumú), úgy a többértelmű lekérdezések további aggregációs és optimalizálási lehetőségeket biztosítanak. Ez a fajta bővítés rekurzív jelleggel megtehető addig a hierarchia szintig, amíg ezen többértelmű lekérdezések a rendszer által kézben tarthatók. Mindehhez a tokenizáción és egy egyszerűbb értelmező logikán kívül nemigen van másra szükség.

2. Intelligencia injektálása a hálózatokba

A hálózati intelligencia egyik kulcsa az értékadás lehetőségének bevezetése, mely szintén minden további nélkül megtehető a mediáció során, az olvasással teljesen analóg módon. Ekkor egy lekérdezés során amennyiben megtaláljuk az adott szolgáltatás Characteristic értékét, úgy először beírjuk a kapott értéket, majd közvetítjük a struktúrát. Példaképpen (hasonlóan C szintaktikát követve) egy „///**(2201=”abc”)**” tartalmú Interest kiadásával, a hálózat összes olyan eszközén, melyen megtalálható a **0x2201**-es UUID-val rendelkező Characteristic érték, abba beírásra kerül az „abc” karaktersor, majd mediációra kerül az azt tartalmazó szolgáltatással egyetemben.

Láthatóan a mediációs folyamat során képesek lehetünk értékek olvasására és írására az eszközökhöz legközelebb eső mediátorok által. A hálózati intelligenciához szükséges következő lépcsőfok egy alkalmas, futási időben végrehajtható parancssorozat (szkript) általunk való összeállításának, és a mediátorokon való értelmezés lehetőségének biztosítása, ami a logika bonyolultságával arányos bonyolultságú értelmezőkkel szintén megvalósítható. Vegyük példaképpen a következő parancssorozatot (az előbbiekkel azonos szintaktika, ezen formában is kerül átadásra):

„{a=”///(0x2201 > 41)?”; /[22 | 23 | 24]//(0x2301=(a ? 1 : 0));}”

Ezen parancssorozat végrehajtása az első mediátor egységen az „a” változónak megfelelő lekérdezéssel kezdődik. A lekérdezésen belül ha egyetlen olyan **0x2201** UUID-val rendelkező Characteristic értéket is találunk a hálózatban, amelynek értéke nagyobb, mint 41, úgy befejezzük a felderítést és közvetítjük az értéket, így annak GATT struktúrája meg fog jelenni a közvetített szolgáltatások között az értékkel

egyetemben. Így az „a” változónak csak referenciával rendelkeznie ezen értékre a lokális struktúrában (amennyiben nem találunk egyezést, úgy üres halmazzal tér vissza lekérdezés, és így a default érték kerül felhasználásra, ami jelen megközelítésben 0). A következő lekérdezésben ezen érték tudatában (behelyettesítés) írunk be 1-es vagy 0-ás értéket a „c” Characteristic a 22-es, 23-as, és 24-es jelzésű eszközökön. Ezt követően azokat is közvetítjük.

Tegyük fel, hogy a **0x2201**-es UUID egy beteg hőmérőjének aktuálisan mért értékéhez tartozik (Celsiusban), a **0x2301**-es pedig egy jelzőfény kapcsolójához. Ekkor ezzel az igen egyszerű parancssorozattal létrehoztunk egy viszonylag egyszerű betegriasztó rendszert, mely, ha egyetlen betegnek is 41 °C fölé megy a testhőmérséklete, úgy a 22-es, 23-as, és 24-es nővérszobákban kigyújt egy-egy jelzőfényt. Azaz alkalmazási szintből „programozva” a hálózatot, sikerült a meditor eszközöket beavatkozásra bírni, ráadásul úgy, hogy a tényleges beavatkozást a jelzőfényekhez legközelebbi eszközök végezték. Tehettük mindezt anélkül, hogy tudnánk, hol helyezkednek el a betegek hőmérői a hálózatban.

Amennyiben valamilyen formában jelöljük, hogy az adott szkript kerüljön perzisztens módon eltárolásra, valamint adott időközönként végrehajtásra (pl. a QoS paraméterrel), akkor beavatkozás és a lekérdezés periodikusan megtörténik, aminek eredményeképpen képesek voltunk intelligenciát injektálni a hálózatba, és ennek megfelelően a továbbiakban nincs is szükség az initiator eszközök jelenlétére, ami komoly előny, hiszen önműködő hálózati alkalmazásokat hozhatunk létre. Különböző szkript templátumok előre definiálásával a rendszer további intelligenciával vértezhető fel, minek során képes lehet szkriptet és lekérdezések generálására is (ez azonban már igen messzire vezet).

A hálózatprogramozás lehetőségével kiegészítve megoldást, igen erős „áthallást” vélhetünk felfedezni a manapság igen nagy kutatói publicitásnak örvendő SDN (Software Defined Networking) hálózati koncepcióval/architektúrával, melyben szintén a hálózatok „programozása” történik [45]. Ugyanakkor megjegyzendő, hogy ott nem igazán az alkalmazásokon, sokkal inkább az azokat kiszolgáló hálózatok optimális működésének (terhelés elosztás, erőforrás kezelés, stb.) elérésén (jól lehet valamelyest az alkalmazások függvényében) van a hangsúly. Ugyanerre, ha felhozzuk a Bluetooth Low Energy technológia működésének fontosabb paramétereit (pl. advInterval, connInterval, scanInterval, ATT_MTU, stb.) a GATT szolgáltatások szintjére, az SMS lehetőséget teremthet erre is, mind elosztott, mind centrális megközelítések mentén (pl. a QoS paraméter segítségével meghatározhatjuk az igényt, s a mediátorok a megfelelő szkriptek végrehajtásával lehetővé teszik azt).

Konklúzió

Az alapfeladat, melyre az SMS kidolgozása során a megoldást kerestem, a GATT funkciók transzparens módon való végrehajtása volt a többes ugrásos Bluetooth Low Energy hálózatokban. Az egyetlen olyan funkció, amely bizonyos értelemben absztrakt megközelítést igényelt, éppen a szolgáltatások felderítése volt, mely az egyik (ha nem az egyetlen) legfontosabb folyamatnak tekinthető az egész GATT keretrendszerben. A megközelítés alapötletét az NDN koncepció adta, mely egy tartalomközpontú hálózati paradigmát követ, s így inkább az információhoz (tartalomhoz) való hozzájutást teszi lehetővé, mintsem csatornát biztosít az egyes állomások adatcseréjéhez. Ezen megközelítés jelen esetben is hasznosnak bizonyult, mindannak ellenére, hogy alapvetően nem ilyen jellegű problémákra kísérel meg választ adni az eredeti koncepció.

A névtér definiálásával, valamint a különböző procedúrák kiötlésével sikerült kidolgoznom egy olyan szolgáltatás mediációs szolgáltatást (SMS), mely általánosan alkalmazható bármilyen szolgáltatás esetén, megfér bármilyen szolgáltatás mellett és jellegéből fakadóan visszirányban kompatibilis bármilyen Bluetooth Low Energy eszközzel. Ugyanakkor számos kérdés (QoS, Timeout, szövevényes topológia, biztonság) megválaszolatlan maradt, annak komplexitásából és a megfelelő elméleti modellek, valamint mérések hiányából fakadóan. Mindezek ellenére a kidolgozott szolgáltatás jelen formában is képes elméletben, egyszerűbb szkenáriók (feszítőfa topológia, csak a kapcsolat ideje alatt élő közvetített szolgáltatások, kevés számú szolgáltatás, stb.) esetén az elvárt funkciókat biztosítani.

Az elméleti működést verifikálandó egy „multi-vendor” kísérleti rendszerben implementáltam és teszteltem a szolgáltatást. Egyes funkciók azonban módosításra, valamint igazításra szorultak ezen gyakorlati megvalósítás során, melynek okai az alkalmazott BLE stack hiányosságaira vezethető vissza. Ugyanakkor ezen módosítások segítségével a szolgáltatást sikerült működésre bírni, s ennek megfelelően annak létjogosultságát igazolni. Bár az eredmények alapvetően bíztatóak (alacsony késleltetés, ami a technológia jellegéből fakad) a megvalósítás rámutatott a rendszer skálázhatóságának korlátaira (memória, teljesítmény) is. Ennek megfelelően egy teljes és minden szempontból általános, rugalmasan módosítható, fejleszthető és jól skálázódó megoldás kidolgozása mindenképpen indokolt.

Noha a megoldásnak alapvetően nem volt célja, olyan új lehetőségek merültek fel a kutatás és a fejlesztés során melyekkel a különböző nézetkialakítási rutinok finomhangolhatóvá válnak (a névtér egy adott részhalmazának reguláris kifejezésekkel és feltételvizsgálatokkal való manipulációján keresztül), valamint elosztott intelligencia injektálható a hálózatba (az értékadás, valamint a szkriptek bevezetésével), azaz a többes ugrásos Bluetooth Low Energy hálózatok programozhatóvá tehetők a meglévő eljárások és funkciók továbbfejlesztésével. Ez alapjában változtathatja meg a többes ugrásos Bluetooth Low Energy hálózatokról alkotott képünket, hiszen egyetlen elosztott, tanítható és intelligens entitásként tekinthetünk rájuk.

Emellett az SMS bármely véges névtérrel definiálható (al)rendszerben működőképes lehet a megfelelő funkciók adott technológiában értelmezhető absztrakciójával. Mindebből következően a megoldás továbbgondolása és különböző szempontokból való további vizsgálata szintén megfontolandó.

Irodalomjegyzék

[1] „Bluetooth Low Energy,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Bluetooth_low_energy. [Hozzáférés dátuma: 13 október 2014].

[2] „Core Specification v4.0,” Bluetooth SIG, 30 június 2010. [Online]. Available: http://www.bluetooth.org/. [Hozzáférés dátuma: 7 március 2012].

[3] „iPhone 4S,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/IPhone_4S. [Hozzáférés dátuma: 13 október 2014].

[4] „MFi Program,” Apple, [Online]. Available: https://developer.apple.com/programs/mfi/. [Hozzáférés dátuma: 13 október 2014].

[5] R. H. S. S. B. Matthew Xie, Szerző, *Google I/O 2013 - Best Practices for Bluetooth Development.* [Performance]. Google Inc., 2013.

[6] „Android 4.3 APIs,” Google Inc., [Online]. Available: https://developer.android.com/about/versions/android-4.3.html. [Hozzáférés dátuma: 13 október 2014].

[7] „Android 4.1.2 release 2.1,” 22 július 2013. [Online]. Available: https://android.googlesource.com/platform/external/bluetooth/bluez/+/android-4.1.2_r2.1. [Hozzáférés dátuma: 13 október 2014].

[8] „Android 4.2.1 release 1,” 12 december 2012. [Online]. Available: https://android.googlesource.com/platform/external/bluetooth/bluedroid/+/android-4.2.1_r1_. [Hozzáférés dátuma: 13 október 2014].

[9] Z. Epstein, „Android just topped iOS in global usage for the first time ever,” BGR Media, 1 augusztus 2014. [Online]. Available: http://bgr.com/2014/08/01/android-ios-market-share-july-2014/. [Hozzáférés dátuma: 13 október 2013].

[10] „Smart Nudge,” Smart Nudge, [Online]. Available: http://www.smartnudge.com.au/. [Hozzáférés dátuma: 13 október 2014].

[11] „H7 heart rate sensor,” Polar, [Online]. Available: http://www.polar.com/us-en/products/accessories/H7_heart_rate_sensor. [Hozzáférés dátuma: 13 október 2013].

[12] „Tempo,” BlueMeastro, [Online]. Available: http://bluemaestro.com/tempo-environment-monitor/. [Hozzáférés dátuma: 13 október 2014].

[13] „Pebble Watch,” Pebble, [Online]. Available: https://getpebble.com/pebble. [Hozzáférés dátuma: 13 október 2014].

[14] „BLUETOOTH LOW ENERGY: BRIDGING THE GAP BETWEEN MOBILE DEVICES AND ALL OTHER KIND OF EMBEDDED DEVICES,” Krasamo, [Online]. Available: http://www.krasamo.com/. [Hozzáférés dátuma: 13 október 2014].

[15] „Internet of Things,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Internet_of_Things. [Hozzáférés dátuma: 13 október 2014].

[16] „Transmission of IPv6 Packets over Bluetooth Low Energy,” 25 október 2011. [Online]. Available: https://datatracker.ietf.org/doc/draft-patil-6lowpan-v6over-btle/. [Hozzáférés dátuma: 13 október 2013].

[17] B. SIG, „Core Version 4.1,” 3 december 2013. [Online]. Available: https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=282159. [Hozzáférés dátuma: 13 október 2014].

[18] J. Decuir, „Communications Society, Bluetooth 4.0: Low Energy,” IEEE, 2010. [Online]. Available: http://chapters.comsoc.org/vancouver/BTLER3.pdf. [Hozzáférés dátuma: 14 október 2014].

[19] K. M. Jouni Tervonen, „Multihop Data Transfer Service for Bluetooth Low Energy,” in *IEEE,* 2013 13th International Conference on ITS Telecommunications (ITST), 2013.

[20] T. Instruments, „Bluetooth low energy software stack and tools,” 12 november 2013. [Online]. Available: http://www.ti.com/tool/ble-stack. [Hozzáférés dátuma: 13 október 2014].

[21] N. Semiconductor, „S110 SoftDevice v7.0,” [Online]. Available: https://www.nordicsemi.com/eng/Products/S110-SoftDevice-v7.0. [Hozzáférés dátuma: 13 október 2014].

[22] „Bluetooth for All!,” 20 február 2014. [Online]. Available: https://code.google.com/p/btstack/. [Hozzáférés dátuma: 13 október 2014].

[23] V. Jacobson, Szerző, *A New Way to look at Networking.* [Performance]. Google Inc., 2006.

[24] Wikipedia, „Content Centric Networking," 14 szeptember 2014. [Online]. Available: http://en.wikipedia.org/wiki/Content_centric_networking. [Hozzáférés dátuma: 11 október 2014].

[25] „Named Data Networking," Wikipedia, 19 szeptember 2014. [Online]. Available: http://en.wikipedia.org/wiki/Named_data_networking. [Hozzáférés dátuma: 11 október 2014].

[26] „Information-Centric Networking Research Group (ICNRG)," IRTF, [Online]. Available: http://trac.tools.ietf.org/group/irtf/trac/wiki/icnrg. [Hozzáférés dátuma: 11 október 2014].

[27] „Information-Centric Networking Research Group (ICNRG)," IRTF, 10 június 2013. [Online]. Available: https://irtf.org/icnrg. [Hozzáférés dátuma: 11 október 2014].

[28] „Named Data Networking," [Online]. Available: http://named-data.net/project/archoverview/. [Hozzáférés dátuma: 11 október 2014].

[29] A. A. J. B. V. J. k. c. P. C. C. P. L. W. B. Z. Lixia Zhang, „Named data networking," *ACM SIGCOMM Computer Communication Review,* %1. kötet44, %1. szám3, pp. 66-73, 2014.

[30] G. T. E. U. L. Z. Paolo Gasti, „DoS & DDoS in Named Data Networking," in *IEEE*, 22nd International Conference on Computer Communications and Networks (ICCCN), 2013.

[31] „NDN Packet Format Specification," [Online]. Available: http://named-data.net/doc/ndn-tlv/. [Hozzáférés dátuma: 13 október 2014].

[32] D. K. S. J. D. T. M. P. N. B. R. B. Van Jacobson, „Networking named content," *Communications of the ACM,* %1. kötet55, %1. szám1, pp. 117-124, 2012.

[33] P. V. M. Tamer Özsu, Principles of Distributed Database Systems, Third Edition, New York, USA: Springer, 2011.

[34] „Bluetooth Assigned Numbers," Bluetooth SIG, [Online]. Available: https://www.bluetooth.org/en-us/specification/assigned-numbers. [Hozzáférés dátuma: 14 október 2014].

[35] „Device Information,” Bluetooth SIG, [Online]. Available: https://developer.bluetooth.org/gatt/services/Pages/ServiceViewer.aspx?u=org.bluetooth.service.device_information.xml. [Hozzáférés dátuma: 15 október 2014].

[36] „cc2541 - 2.4-GHz Bluetooth® low energy and Proprietary System-on-Chip,” Texas Instruments, 2012. [Online]. Available: http://www.ti.com/product/cc2541. [Hozzáférés dátuma: 15 október 2014].

[37] „Services,” Bluetooth SIG, [Online]. Available: https://developer.bluetooth.org/gatt/services/Pages/ServicesHome.aspx. [Hozzáférés dátuma: 15 október 2014].

[38] „CC2540 Mini Development Kit,” Texas Instruments, [Online]. Available: http://www.ti.com/tool/cc2540dk-mini. [Hozzáférés dátuma: 15 október 2014].

[39] „nRF51822 Development Kit,” Nordic Semiconductors, [Online]. Available: https://www.nordicsemi.com/eng/Products/Bluetooth-Smart-Bluetooth-low-energy/nRF51822-Development-Kit. [Hozzáférés dátuma: 15 október 2014].

[40] „S130 SoftDevice,” Nordic Semiconductors, [Online]. Available: https://www.nordicsemi.com/eng/Products/S130-SoftDevice. [Hozzáférés dátuma: 16 október 2014].

[41] „Realterm: Serial Terminal,” [Online]. Available: monitorozandó . [Hozzáférés dátuma: 15 október 2014].

[42] „BACnet,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/BACnet. [Hozzáférés dátuma: 16 október 2014].

[43] „KNX (standard),” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/KNX_(standard). [Hozzáférés dátuma: 16 október 2014].

[44] „LonWorks,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/LonWorks. [Hozzáférés dátuma: 16 október 2014].

[45] „Sofware-defined Networking,” Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/Software-defined_networking. [Hozzáférés dátuma: 16 október 2014].

Függelék

1. Rövidítésjegyzék

A-FHSS	Adaptive Frequency Hopping Spread Spectrum
AMP	Alternate MAC/PHY
API	Application Programming Interface
ACL	Asynchronous Connection Logical Transport
ATT	Attribute Protocol
BLE	Bluetooth Low Energy
BGP	Border Gateway Protocol
CCCD	Client Characteristic Configuration Descriptor
CCN	Content Centric Networking
CS	Content Store
DNS	Domain Name System
FIB	Forward Information Base
FDMA	Frequency Division Multiple Access
GFSK	Gaussian Frequency Shift Keying
GATT	Generic Attribute Framework
ISM	Industrial, Scientific and Medical
ICN	Information Centric Networking
IS-IS	Intermediate System to Intermediate System
IETF	Internet Engineering Task Force
IP	Internet Protocol
IRTF	Internet Research Task Force
L2CAP	Logical Link Control and Adaptation Protocol
MTU	Maximum Transmission Unit
MAC	Medium Access Control

NDN	Named Data Networking
NSF	National Science Foundation
NGN	New Generation Network
NLOS	Not Line of Sight
OSPF	Open Shortest Path First
OSI	Open System Interconnection
PARC	Palo Alto Research Center
PIT	Pending Interest Table
PHY	Physicial Layer
PoC	Proof of Concept
PDU	Protocol Data Unit
QoS	Quality of Service
RIP	Routing Information Protocol
SMP	Security Management Protocol
SMS	Service Mediation Service
SDN	Software Defined Networking
SIG	Special Interest Group
SoC	Sysmtem on Chip
TDMA	Time Division Multiple Access
TDM	Time Division Multiplex
URI	Uniform Resource Identifier
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UUID	Universally Unique Identifier

2. Kódrészletek

2.1. Struktúrák a felderítéshez és mediációhoz

```
#define MAX_SERVICE_NUM                                    6
#define MAX_CHAR_NUM                                       4
#define MAX_CHAR_DESCRIPTOR_NUM                            3

// Structures used for discovery and mediation

typedef struct {
      uint16_t uuid;
      uint16_t ext_handle;
      uint16_t local_handle;
} desc_t;

typedef struct {
      uint16_t uuid;
      uint16_t local_value_handle;
      ble_gap_addr_t* source;
      uint16_t ext_value_handle;
      uint8_t props;
      uint8_t desc_count;
      desc_t descriptor[MAX_CHAR_DESCRIPTOR_NUM];
      desc_t ndn_desc;
} char_t;

typedef struct {
      uint16_t uuid;
      fib_t* p_fib;
      uint16_t local_handle;
      uint16_t ext_handle;
      uint8_t char_count;
      char_t characteristics[MAX_CHAR_NUM];
} mediated_service_t;
```

3. Eseménynapló-részletek

3.1. SMS felderítés kezdeményezés és válasz

1889;17:13:56.8785 [480111362][ScriptMethodCallThread] WriteRequest sent to handle 0x0013 with value [47, 47, 49, 56, 48, 70, 47]

1890;17:13:56.8845 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:0A-00-04-00-12-13-00-2F-2F-31-38-30-46-2F

1891;17:13:56.8855 [480111362][ScriptMethodCallThread] Sending packet: AttWriteRequest, 12-13-00-2F-2F-31-38-30-46-2F, Handle: 0x0013, Value: 2F-2F-31-38-30-46-2F

1892;17:13:56.8855 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-0E-00-0A-00-04-00-12-13-00-2F-2F-31-38-30-46-2F

1893;17:13:56.8855 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

1894;17:13:56.8855 [480111362][ScriptMethodCallThread] Number of data packets sent: 117

1895;17:13:56.8905 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

1896;17:13:56.8905 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

1897;17:13:56.8905 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

1898;17:13:56.9015 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-05-00-01-00-04-00-13

1899;17:13:56.9015 [480111362][ReadPacketQueueThread] Received packet: AttWriteResponse, 13

1900;17:13:56.9105 [480111362][ScriptMethodCallThread] Updated handle 0013 with value [47, 47, 49, 56, 48, 70, 47]

1901;17:13:56.9125 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1D-10-00-01

1902;17:13:56.9125 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueIndication, 1D-10-00-01, Handle: 0x0010, Value: 01

1903;17:13:56.9135 [480111362][ScriptThread] WRITE DATA PACKET:01-00-04-00-1E

1904;17:13:56.9135 [480111362][ScriptThread] Sending packet: AttHandleValueConfirmation, 1E

1905;17:13:56.9145 [480111362][ScriptThread] Serial port write: 02-00-00-05-00-01-00-04-00-1E

1906;17:13:56.9145 [480111362][ScriptThread] Data buffer queue length (+1): 1

1907;17:13:56.9145 [480111362][ScriptThread] Number of data packets sent: 118

1908;17:13:56.9155 [480111362][ScriptThread] Received a HandleValueIndication on handle 0010 with value 01

1909;17:13:56.9205 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

1910;17:13:56.9205 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

1911;17:13:56.9205 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

1912;17:14:04.9310 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1D-10-00-00

1913;17:14:04.9310 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueIndication, 1D-10-00-00, Handle: 0x0010, Value: 00

1914;17:14:04.9320 [480111362][ScriptThread] WRITE DATA PACKET:01-00-04-00-1E

1915;17:14:04.9320 [480111362][ScriptThread] Sending packet: AttHandleValueConfirmation, 1E

1916;17:14:04.9320 [480111362][ScriptThread] Serial port write: 02-00-00-05-00-01-00-04-00-1E

1917;17:14:04.9320 [480111362][ScriptThread] Data buffer queue length (+1): 1

1918;17:14:04.9320 [480111362][ScriptThread] Number of data packets sent: 119

1919;17:14:04.9340 [480111362][ScriptThread] Received a HandleValueIndication on handle 0010 with value 00

1920;17:14:04.9450 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

1921;17:14:04.9450 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

1922;17:14:04.9450 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3.2. SMS GATT olvasás

3365;17:25:26.0580 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-19-00

3366;17:25:26.0580 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-19-00, Handle: 0x0019

3367;17:25:26.0580 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-19-00

3368;17:25:26.0580 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3369;17:25:26.0580 [480111362][ScriptMethodCallThread] Number of data packets sent: 234

3370;17:25:26.0690 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3371;17:25:26.0690 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3372;17:25:26.0690 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3373;17:25:26.1480 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-5F

3374;17:25:26.1490 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-5F, AttributeValue: 5F

3375;17:25:26.1970 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0019, value (0x): 5F

3376;17:25:26.9140 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-19-00

3377;17:25:26.9140 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-19-00, Handle: 0x0019

3378;17:25:26.9140 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-19-00

3379;17:25:26.9140 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3380;17:25:26.9140 [480111362][ScriptMethodCallThread] Number of data packets sent: 235

3381;17:25:26.9290 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3382;17:25:26.9290 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3383;17:25:26.9290 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3384;17:25:27.0090 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-5E

3385;17:25:27.0090 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-5E, AttributeValue: 5E

3386;17:25:27.0430 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0019, value (0x): 5E

3387;17:25:27.8181 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-19-00

3388;17:25:27.8181 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-19-00, Handle: 0x0019

3389;17:25:27.8181 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-19-00

3390;17:25:27.8181 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3391;17:25:27.8181 [480111362][ScriptMethodCallThread] Number of data packets sent: 236

3392;17:25:27.8291 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3393;17:25:27.8291 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3394;17:25:27.8291 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3395;17:25:27.9091 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-5E

3396;17:25:27.9091 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-5E, AttributeValue: 5E

3397;17:25:27.9471 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0019, value (0x): 5E

3398;17:25:29.7142 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-1D-00

3399;17:25:29.7142 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-1D-00, Handle: 0x001D

3400;17:25:29.7142 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-1D-00

3401;17:25:29.7142 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3402;17:25:29.7142 [480111362][ScriptMethodCallThread] Number of data packets sent: 237

3403;17:25:29.7282 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3404;17:25:29.7282 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3405;17:25:29.7282 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3406;17:25:29.8082 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-4E

3407;17:25:29.8082 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-4E, AttributeValue: 4E

3408;17:25:29.8432 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x001D, value (0x): 4E

3409;17:25:30.2662 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-1D-00

3410;17:25:30.2662 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-1D-00, Handle: 0x001D

3411;17:25:30.2662 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-1D-00

3412;17:25:30.2662 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3413;17:25:30.2662 [480111362][ScriptMethodCallThread] Number of data packets sent: 238

3414;17:25:30.3082 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3415;17:25:30.3082 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3416;17:25:30.3082 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3417;17:25:30.4092 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-4E

3418;17:25:30.4092 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-4E, AttributeValue: 4E

3419;17:25:30.4432 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x001D, value (0x): 4E

3420;17:25:30.8102 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-1D-00

3421;17:25:30.8102 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-1D-00, Handle: 0x001D

3422;17:25:30.8102 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-1D-00

3423;17:25:30.8102 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3424;17:25:30.8102 [480111362][ScriptMethodCallThread] Number of data packets sent: 239

3425;17:25:30.8272 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3426;17:25:30.8282 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3427;17:25:30.8282 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3428;17:25:30.9082 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-4E

3429;17:25:30.9082 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-4E, AttributeValue: 4E

3430;17:25:30.9162 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x001D, value (0x): 4E

3431;17:25:33.3374 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-22-00

3432;17:25:33.3384 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-22-00, Handle: 0x0022

3433;17:25:33.3384 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-22-00

3434;17:25:33.3384 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3435;17:25:33.3384 [480111362][ScriptMethodCallThread] Number of data packets sent: 240

3436;17:25:33.3474 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3437;17:25:33.3484 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3438;17:25:33.3484 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3439;17:25:33.5084 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-5B

3440;17:25:33.5094 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-5B, AttributeValue: 5B

3441;17:25:33.5194 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0022, value (0x): 5B

3442;17:25:33.9384 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-22-00

3443;17:25:33.9384 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-22-00, Handle: 0x0022

3444;17:25:33.9394 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-22-00

3445;17:25:33.9394 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3446;17:25:33.9394 [480111362][ScriptMethodCallThread] Number of data packets sent: 241

3447;17:25:33.9484 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3448;17:25:33.9484 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3449;17:25:33.9484 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3450;17:25:34.1084 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-5B

3451;17:25:34.1084 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-5B, AttributeValue: 5B

3452;17:25:34.1174 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0022, value (0x): 5B

3453;17:25:34.5774 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-22-00

3454;17:25:34.5774 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-22-00, Handle: 0x0022

3455;17:25:34.5784 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-22-00

3456;17:25:34.5784 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3457;17:25:34.5784 [480111362][ScriptMethodCallThread] Number of data packets sent: 242

3458;17:25:34.6074 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3459;17:25:34.6074 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3460;17:25:34.6074 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3461;17:25:34.7685 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-5B

3462;17:25:34.7685 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-5B, AttributeValue: 5B

3463;17:25:34.7725 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0022, value (0x): 5B

3464;17:25:36.6576 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-26-00

3465;17:25:36.6576 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-26-00, Handle: 0x0026

3466;17:25:36.6586 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-26-00

3467;17:25:36.6586 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3468;17:25:36.6586 [480111362][ScriptMethodCallThread] Number of data packets sent: 243

3469;17:25:36.6686 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3470;17:25:36.6686 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3471;17:25:36.6686 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3472;17:25:36.8096 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-54

3473;17:25:36.8096 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-54, AttributeValue: 54

3474;17:25:36.8376 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0026, value (0x): 54

3475;17:25:37.3356 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-26-00

3476;17:25:37.3356 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-26-00, Handle: 0x0026

3477;17:25:37.3366 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-26-00

3478;17:25:37.3366 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3479;17:25:37.3366 [480111362][ScriptMethodCallThread] Number of data packets sent: 244

3480;17:25:37.3476 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3481;17:25:37.3476 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3482;17:25:37.3476 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3483;17:25:37.5086 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-54

3484;17:25:37.5086 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-54, AttributeValue: 54

3485;17:25:37.5146 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0026, value (0x): 54

3486;17:25:37.9986 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:03-00-04-00-0A-26-00

3487;17:25:37.9996 [480111362][ScriptMethodCallThread] Sending packet: AttReadRequest, 0A-26-00, Handle: 0x0026

3488;17:25:37.9996 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-07-00-03-00-04-00-0A-26-00

3489;17:25:37.9996 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

3490;17:25:37.9996 [480111362][ScriptMethodCallThread] Number of data packets sent: 245

3491;17:25:38.0086 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

3492;17:25:38.0086 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

3493;17:25:38.0086 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

3494;17:25:38.1686 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-06-00-02-00-04-00-0B-54

3495;17:25:38.1696 [480111362][ReadPacketQueueThread] Received packet: AttReadResponse, 0B-54, AttributeValue: 54

3496;17:25:38.1856 [480111362][ScriptMethodCallThread] Received Read Response, handle: 0x0026, value (0x): 54

3.3. SMS GATT írás és notifikációk

24426;18:17:05.3562 [480111362][MainThread] EnableServices({0x000B:2,0x0011:2,0x001A:1,0x001E:1,})

24427;18:17:05.3722 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:05-00-04-00-12-1E-00-01-00

24428;18:17:05.3722 [480111362][ScriptMethodCallThread] Sending packet: AttWriteRequest, 12-1E-00-01-00, Handle: 0x001E, Value: 01-00

24429;18:17:05.3732 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-09-00-05-00-04-00-12-1E-00-01-00

24430;18:17:05.3732 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

24431;18:17:05.3732 [480111362][ScriptMethodCallThread] Number of data packets sent: 41

24432;18:17:05.3832 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

24433;18:17:05.3832 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

24434;18:17:05.3832 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

24435;18:17:05.4032 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-05-00-01-00-04-00-13

24436;18:17:05.4032 [480111362][ReadPacketQueueThread] Received packet: AttWriteResponse, 13

24437;18:17:05.4242 [480111362][ScriptMethodCallThread] Updated handle 001E with value [1, 0]

24438;18:17:05.4262 [480111362][ScriptMethodCallThread] Successfully updated the store value of CCCD

24439;18:17:05.4272 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:05-00-04-00-12-11-00-02-00

24440;18:17:05.4272 [480111362][ScriptMethodCallThread] Sending packet: AttWriteRequest, 12-11-00-02-00, Handle: 0x0011, Value: 02-00

24441;18:17:05.4272 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-09-00-05-00-04-00-12-11-00-02-00

24442;18:17:05.4272 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

24443;18:17:05.4272 [480111362][ScriptMethodCallThread] Number of data packets sent: 42

24444;18:17:05.4632 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

24445;18:17:05.4632 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

24446;18:17:05.4632 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

24447;18:17:05.4842 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-05-00-01-00-04-00-13

24448;18:17:05.4842 [480111362][ReadPacketQueueThread] Received packet: AttWriteResponse, 13

24449;18:17:05.4902 [480111362][ScriptMethodCallThread] Updated handle 0011 with value [2, 0]

24450;18:17:05.4902 [480111362][ScriptMethodCallThread] Successfully updated the store value of CCCD

24451;18:17:05.4902 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:05-00-04-00-12-0B-00-02-00

24452;18:17:05.4902 [480111362][ScriptMethodCallThread] Sending packet: AttWriteRequest, 12-0B-00-02-00, Handle: 0x000B, Value: 02-00

24453;18:17:05.4912 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-09-00-05-00-04-00-12-0B-00-02-00

24454;18:17:05.4912 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

24455;18:17:05.4912 [480111362][ScriptMethodCallThread] Number of data packets sent: 43

24456;18:17:05.5042 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-10-00-0C-00-05-00-12-02-08-00-08-00-08-00-00-00-90-01

24457;18:17:05.5042 [480111362][ReadPacketQueueThread] Received packet: L2capConnectionParameterUpdateRequest, 12-02-08-00-08-00-08-00-00-00-90-01, IntervalMin: 10ms, IntervalMax: 10ms,

SlaveLatency: 0, Timeout: 4000ms

24458;18:17:05.5082 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

24459;18:17:05.5082 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

24460;18:17:05.5082 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

24461;18:17:05.5092 [480111362][ScriptThread] Received Connection Parameter Update Request

24462;18:17:05.5112 [480111362][ScriptThread] launching connection update worker

24463;18:17:05.5182 [480111362][] WRITE DATA PACKET:06-00-05-00-13-02-02-00-00-00

24464;18:17:05.5182 [480111362][] Sending packet: L2capConnectionParameterUpdateResponse, 13-02-02-00-00-00, Result: ConnectionParametersAccepted

24465;18:17:05.5182 [480111362][] Serial port write: 02-00-00-0A-00-06-00-05-00-13-02-02-00-00-00

24466;18:17:05.5182 [480111362][] Data buffer queue length (+1): 1

24467;18:17:05.5182 [480111362][] Number of data packets sent: 44

24468;18:17:05.5192 [480111362][] ConnectionParameterUpdateResponse sent with ACCEPTED response

24469;18:17:05.5252 [480111362][] HCI command: BTLE_CMD_LE_CONNECTION_UPDATE, ConnectionHandle: 0x0000, ConnIntervalMin: 10ms, ConnIntervalMax: 10ms, ConnLatency: 0, SupervisionTimeout: 4000ms,

MinCeLength: 2, MaxCeLength: 2

24470;18:17:05.5272 [480111362][] Serial port write: 01-13-20-0E-00-00-08-00-08-00-00-00-90-01-02-00-02-00

24471;18:17:05.5282 [480111362][] Connection Parameters Update sent. ConnInterval:10.0ms, SlaveLatency:0, SupervisionTimeout:4000.0ms

24472;18:17:05.5282 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-05-00-01-00-04-00-13

24473;18:17:05.5282 [480111362][ReadPacketQueueThread] Received packet: AttWriteResponse, 13

24474;18:17:05.5282 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

24475;18:17:05.5282 [480111362][ScriptMethodCallThread] Updated handle 000B with value [2, 0]

24476;18:17:05.5292 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

24477;18:17:05.5292 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

24478;18:17:05.5292 [480111362][ReadPacketQueueThread] Serial port read: 04-0F-04-00-01-13-20

24479;18:17:05.5292 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_COMMAND_STATUS, OpCode: BTLE_CMD_LE_CONNECTION_UPDATE, Status: BTLE_STATUS_CODE_SUCCESS

24480;18:17:05.5352 [480111362][ScriptMethodCallThread] Successfully updated the store value of CCCD

24481;18:17:05.5362 [480111362][ScriptMethodCallThread] WRITE DATA PACKET:05-00-04-00-12-1A-00-01-00

24482;18:17:05.5362 [480111362][ScriptMethodCallThread] Sending packet: AttWriteRequest, 12-1A-00-01-00, Handle: 0x001A, Value: 01-00

24483;18:17:05.5362 [480111362][ScriptMethodCallThread] Serial port write: 02-00-00-09-00-05-00-04-00-12-1A-00-01-00

24484;18:17:05.5362 [480111362][ScriptMethodCallThread] Data buffer queue length (+1): 1

24485;18:17:05.5362 [480111362][ScriptMethodCallThread] Number of data packets sent: 45

24486;18:17:05.5642 [480111362][ReadPacketQueueThread] Serial port read: 04-13-05-01-00-00-01-00

24487;18:17:05.5642 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_NUMBER_OF_COMPLETED_PACKETS, NumberOfHandles: 1, ConnectionHandle: 0x0000, NumberOfPackets: 1

24488;18:17:05.5642 [480111362][ReadPacketQueueThread] Data buffer queue length: 0

24489;18:17:05.6032 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-05-00-01-00-04-00-13

24490;18:17:05.6032 [480111362][ReadPacketQueueThread] Received packet: AttWriteResponse, 13

24491;18:17:05.6302 [480111362][ScriptMethodCallThread] Updated handle 001A with value [1, 0]

24492;18:17:05.6322 [480111362][ScriptMethodCallThread] Successfully updated the store value of CCCD

24493;18:17:05.6832 [480111362][ReadPacketQueueThread] Serial port read: 04-3E-0A-03-00-00-00-08-00-00-00-90-01

24494;18:17:05.6832 [480111362][ReadPacketQueueThread] HCI event: BTLE_EVENT_LE_CONNECTION_UPDATE_COMPLETE, ConnectionHandle: 0x0, ConnectionInterval: 10ms, Latency: 0, SupervisionTimeout: 4000ms

24495;18:17:05.6942 [480111362][] BackgroundWorker completed

24496;18:17:07.2533 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-5D

24497;18:17:07.2533 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-5D, Handle: 0x0019, Value: 5D

24498;18:17:07.2703 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 5D

24499;18:17:07.4033 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-5D

24500;18:17:07.4033 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-5D, Handle: 0x001D, Value: 5D

24501;18:17:07.4053 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 5D

24502;18:17:09.2534 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-5C

24503;18:17:09.2534 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-5C, Handle: 0x0019, Value: 5C

24504;18:17:09.2554 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 5C

24505;18:17:09.4045 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-5C

24506;18:17:09.4045 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-5C, Handle: 0x001D, Value: 5C

24507;18:17:09.4055 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 5C

24508;18:17:11.2546 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-5B

24509;18:17:11.2546 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-5B, Handle: 0x0019, Value: 5B

24510;18:17:11.2566 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 5B

24511;18:17:11.4046 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-5B

24512;18:17:11.4046 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-5B, Handle: 0x001D, Value: 5B

24513;18:17:11.4056 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 5B

24514;18:17:13.2537 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-5A

24515;18:17:13.2547 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-5A, Handle: 0x0019, Value: 5A

24516;18:17:13.2567 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 5A

24517;18:17:13.4037 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-5A

24518;18:17:13.4037 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-5A, Handle: 0x001D, Value: 5A

24519;18:17:13.4047 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 5A

24520;18:17:15.3038 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-59

24521;18:17:15.3038 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-59, Handle: 0x0019, Value: 59

24522;18:17:15.3058 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 59

24523;18:17:15.4038 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-59

24524;18:17:15.4038 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-59, Handle: 0x001D, Value: 59

24525;18:17:15.4038 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 59

24526;18:17:17.2539 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-58

24527;18:17:17.2539 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-58, Handle: 0x0019, Value: 58

24528;18:17:17.2549 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 58

24529;18:17:17.4059 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-58

24530;18:17:17.4059 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-58, Handle: 0x001D, Value: 58

24531;18:17:17.4079 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 58

24532;18:17:19.2540 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-57

24533;18:17:19.2540 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-57, Handle: 0x0019, Value: 57

24534;18:17:19.2580 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 57

24535;18:17:19.4040 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-57

24536;18:17:19.4040 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-57, Handle: 0x001D, Value: 57

24537;18:17:19.4050 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 57

24538;18:17:21.3041 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-56

24539;18:17:21.3041 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-56, Handle: 0x0019, Value: 56

24540;18:17:21.3061 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 56

24541;18:17:21.4531 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-56

24542;18:17:21.4531 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-56, Handle: 0x001D, Value: 56

24543;18:17:21.4551 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 56

24544;18:17:23.2532 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-55

24545;18:17:23.2532 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-55, Handle: 0x0019, Value: 55

24546;18:17:23.2552 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 55

24547;18:17:23.4033 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-55

24548;18:17:23.4033 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-55, Handle: 0x001D, Value: 55

24549;18:17:23.4043 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 55

24550;18:17:25.2634 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-54

24551;18:17:25.2644 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-54, Handle: 0x0019, Value: 54

24552;18:17:25.2654 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 54

24553;18:17:25.4034 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-54

24554;18:17:25.4034 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-54, Handle: 0x001D, Value: 54

24555;18:17:25.4044 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 54

24556;18:17:27.2535 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-53

24557;18:17:27.2535 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-53, Handle: 0x0019, Value: 53

24558;18:17:27.2585 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 53

24559;18:17:27.4145 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-53

24560;18:17:27.4145 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-53, Handle: 0x001D, Value: 53

24561;18:17:27.4215 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 53

24562;18:17:29.2546 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-52

24563;18:17:29.2546 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-52, Handle: 0x0019, Value: 52

24564;18:17:29.2566 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 52

24565;18:17:29.4036 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-52

24566;18:17:29.4036 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-52, Handle: 0x001D, Value: 52

24567;18:17:29.4046 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 52

24568;18:17:31.2537 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-51

24569;18:17:31.2537 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-51, Handle: 0x0019, Value: 51

24570;18:17:31.2547 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 51

24571;18:17:31.4037 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-51

24572;18:17:31.4037 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-51, Handle: 0x001D, Value: 51

24573;18:17:31.4067 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 51

24574;18:17:33.2538 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-52

24575;18:17:33.2538 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-52, Handle: 0x0019, Value: 52

24576;18:17:33.2548 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 52

24577;18:17:33.4538 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-52

24578;18:17:33.4538 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-52, Handle: 0x001D, Value: 52

24579;18:17:33.4558 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 52

24580;18:17:35.2539 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-53

24581;18:17:35.2539 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-53, Handle: 0x0019, Value: 53

24582;18:17:35.2559 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 53

24583;18:17:35.4029 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-53

24584;18:17:35.4039 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-53, Handle: 0x001D, Value: 53

24585;18:17:35.4059 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 53

24586;18:17:37.2540 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-19-00-54

24587;18:17:37.2540 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-19-00-54, Handle: 0x0019, Value: 54

24588;18:17:37.2590 [480111362][ScriptThread] Received a HandleValueNotification on handle 0019 with value 54

24589;18:17:37.4041 [480111362][ReadPacketQueueThread] Serial port read: 02-00-20-08-00-04-00-04-00-1B-1D-00-54

24590;18:17:37.4041 [480111362][ReadPacketQueueThread] Received packet: AttHandleValueNotification, 1B-1D-00-54, Handle: 0x001D, Value: 54

24591;18:17:37.4051 [480111362][ScriptThread] Received a HandleValueNotification on handle 001D with value 54

24592;18:17:38.4591 [480111362][MainThread]

DisableServices({0x000B:2,0x0011:2,0x001A:1,0x001E:1,}) féle