Kommunikáció Bluetooth Low Energy alapú piconetek között

TDK dolgozat

Készítette: Balogh András

Tartalomjegyzék

Összefoglaló

A Bluetooth napjaink egyik legelterjedtebb vezeték nélküli hálózati technológiája. A 2010-ben adoptált Bluetooth v4.0 szabvány egy újabb megoldással egészült ki, mely robosztusabb, kisebb komplexitású és sokkal alacsonyabb fogyasztással bír a korábbi változatnál. A szabvány szerinti elnevezés a Bluetooth Low Energy, de egyes korábbi terminológiákban Bluetooth Smart néven is emlegetik. A Wibree technológia - amelyen a Bluetooth Low Energy alapszik - már korábban is ismert volt. Ennek egy valamelyest átdolgozott formája került az említett szabványba. [1]

A Bluetooth Low Energy piconet topológiát valósít meg, azaz egy master és több slave egység kommunikációját teszi lehetővé, a master egység vezérlésének megfelelően. A specifikációban leírtak szerint a technológia nem támogatja a scatternetek, azaz több, összekapcsolt piconetből álló hálózatok létrehozását, minek folytán az ad-hoc jellegű több ugrást igénylő útvonalak nem hozhatók létre az egyes piconetek szétesése nélkül. Ebből következően a kommunikáció csak egy adott node hatósugarában lehetséges. Ezt a hátrányt több, konkurens technológiákat felsorakoztató összehasonlításban is kiemelik. [2] [3]

Bár a specifikáció szigorúan szabályozza az egyes állapotokat, lehetséges olyan kombináció alkalmazása, amely - bár nem teremt teljes értékű kapcsolatot - mégis lehetővé teszi a kommunikációt a piconetek között.

Jelen dolgozatban bemutatásra kerül a technológia, azt követően kifejtésre kerül a megoldás. Végül a valós kísérlet eredményeit tárgyalom. A piconetek közötti átjárás lehetőségével a lefedhető terület megnő, azaz nagyobb kiterjedésű és együttműködő szenzor és adathálózatok hozhatók létre anélkül, hogy egyéb technológia is alkalmazásra kerülne.

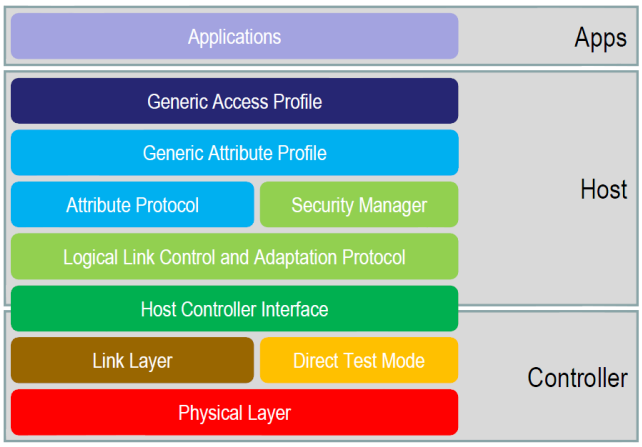
A Bluetooth Low Energy technológia áttekintése

A Bluetooth LE rendszert - a legtöbb vezeték nélküli adatátviteli technológiával szemben - nem a „burst” jellegű, szekvenciális adatcsomagok átviteléhez (relatíve nagyobb fájlok mozgatásához) tervezték, hanem olyan környezetekbe (pl. Szenzorhálózatok - WSN), ahol csak kisebb státuszüzenetek kerülnek átküldésre. [1]

A technológia kulcsfontosságú szerepet kapott a jövő IPv6 alapú, alacsony fogyasztású személyi hálózatainak (6LoWPAN - IETF) sorában, így várhatóan egyre több készülékben fog megjelenni. Az IPv6 a jelentősen megnövelt (128 bit) címterével az „ubiquitous internet”, avagy másik nevén az „Internet of things” jelenséget segíti elő, amelyben minden egyes eszköz (tágabb értelemben entitás) egyedi címmel rendelkezik, és annak használatával a világ bármely pontjáról egy másik eszköz el tudja azt érni. [1] [5]

A következő pontokban bemutatásra kerülnek a Bluetooth LE (Low Energy) rendszer egyes rétegei és azok működése. Azt követve a fontosabb felhasználási területeket és a mobil platformokon való elérhetőséget fejtem ki röviden.

Architekturális áttekintés

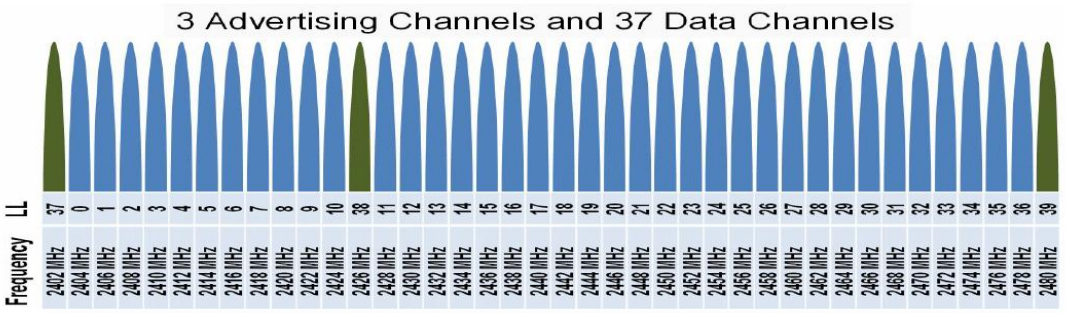


1. ábra [1]

A Bluetooth LE stack rétegei (1. ábra) alapvetően 3 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 (TI, CSR, Atheros stb.), amelyek egyazon integrált áramköri lapkán valósítják meg az összes réteget. [2]

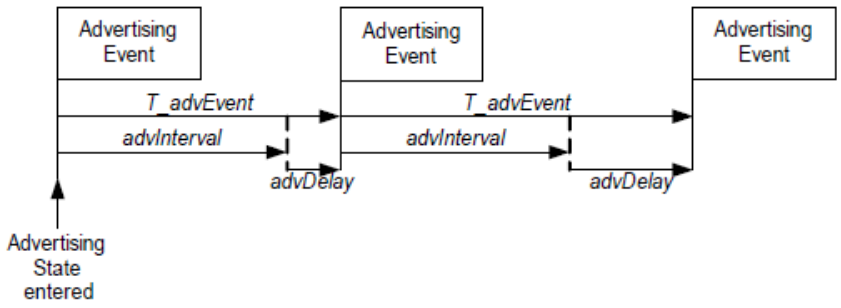
A fizikai réteg (PHY)

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]



2. ábra [1]

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]



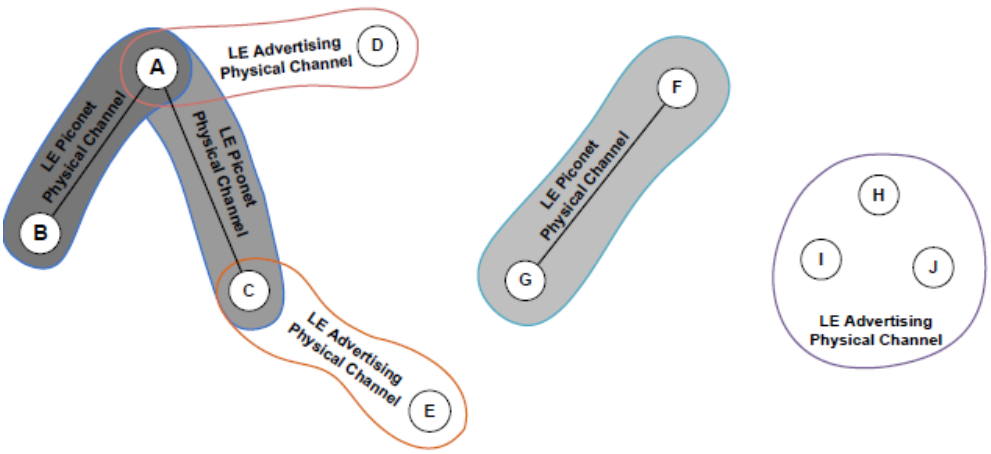
3. ábra [2]

Példaképpen, több egymást követő „Advertising event” esetén (3. ábra) az időbeni megkötések a következőképpen alakulnak: [2]

- $20ms < advInterval < 10{,}24s$ (tetszőlegesen állítható)
- $0ms < advDelay < 10ms$ (pszeudorandom érték)

A Link Layer (MAC)

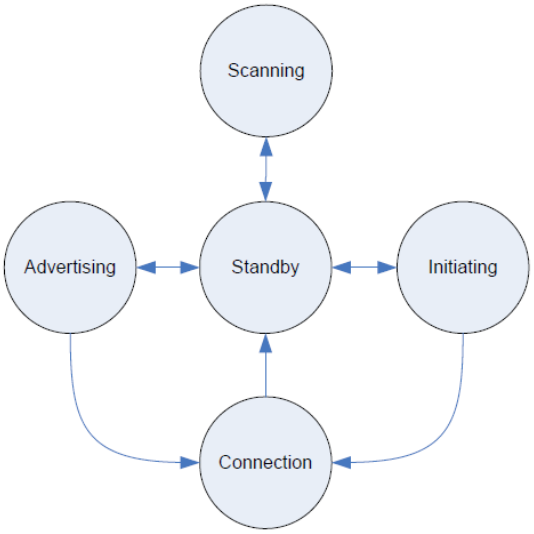
A fizikai csatorna (PHY) felett a réteghierarchiában a fizikai kapcsolatok (MAC) és további felsőbb szintű csatornák és az ezekhez tartozó vezérlő eljárások találhatók. Egy fizikai csatorna csak egy kitüntetett eszköz, a „master”, és egy vagy több „slave” eszköz között lehet (piconet topológia). A direkt kapcsolatok a „slave”-ek között nem lehetségesek, azaz jelen esetben egy „slave”-nek csak egyetlen kapcsolata lehet, és nem is tölthet be párhuzamosan „master” és „slave” szerepet (a korábbi Bluetooth technológiában ez megengedett volt). Ez a korlát alapjában véve nehezíti meg a „piconet”-ek közötti átjárást. A következő (4.) ábra néhány lehetséges szituációt mutat be. [2]



4. ábra [6]

Az **A** és **F** eszközök jelen esetben, egy-egy piconet-ben a master szerepét töltik be. A **B**, **C** és **G** eszközök pedig slave szerepben vesznek részt az adatcsatornákon történő kommunikációban. Ugyanakkor az **A-D** eszközök, illetve a **C-E** eszközök, továbbá a **H-I-J** eszközhármasok között, csoportonként a hirdetési csatornákon zajlik a kommunikáció.

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. [2]



5. ábra [2]

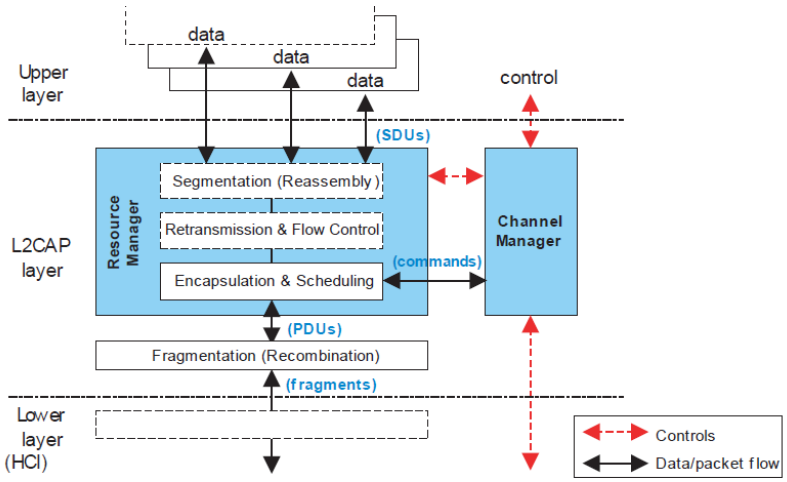
A Link Layer működése egy állapotgéppel írható le (5. ábra). Connection állapotban. Az egyes állapotokban a fizikai kapcsolati réteg (Link Layer) a következő feladatokat látja el:

- **Standby** állapotban a Link Layer se nem küld, se 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.

A Link Layerben hajtódnak végre a címzéssel kapcsolatos mechanizmusok is. Egy Bluetooth LE modulnak 2 féle címe lehet: Public és Random. A Public címek megfelelnek a szokásos Bluetooth MAC címeknek, míg a Random címek egy adott (korábbi) kapcsolathoz generálódhatnak, így kizárva az esetleges támadásokat. [2]

Az L2CAP réteg



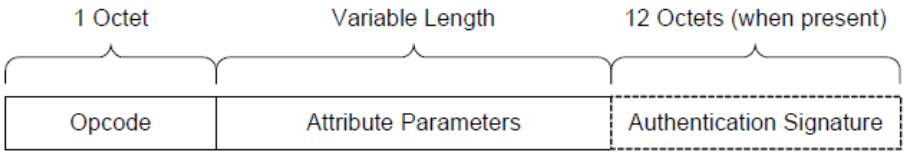
6. ábra [6]

A Link Layer fölött helyezkedik el az L2CAP (Logical Link Control Adaptation Protocol), amely csatorna alapú absztrakciót lát el a szolgáltatások és alkalmazások számára (6. ábra). 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]

Az ATT és SMP protokollok

Az L2CAP fölött helyezkednek el, egymás mellett az Attribute Protocol (ATT) és a Security Manager Protocol (SMP). Az utóbbi fix L2CAP csatornát használ a biztonsági szolgáltatások nyújtására (párosítás, kulcskezelés, hash generálás stb.), míg 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]

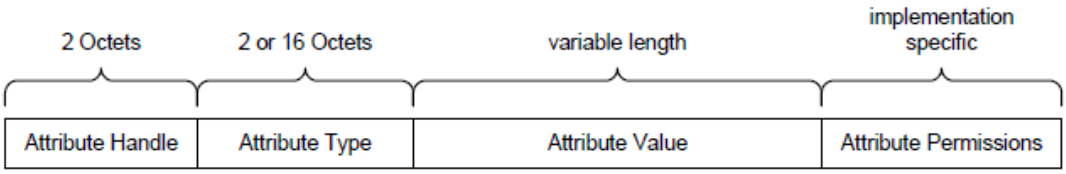
Az ATT a következő PDU (Protocol Data Unit) formátumot határozza meg (7. ábra):



7. ábra [2]

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:

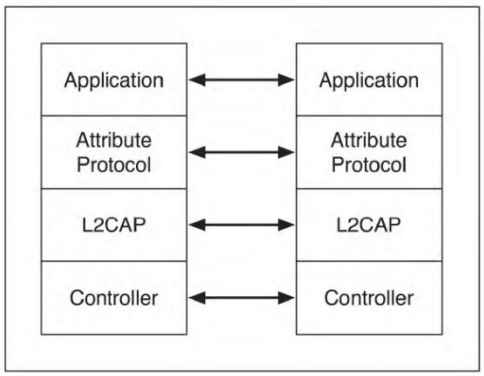


8. ábra [2]

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 kitüntett *Attribute*-ra nézve. [2]

Az *Attribute*-ok lényegére, illetve jelentésére a következő réteg, a GATT réteg kifejtése során derül fény. [2]

A GATT keretrendszer



9. ábra

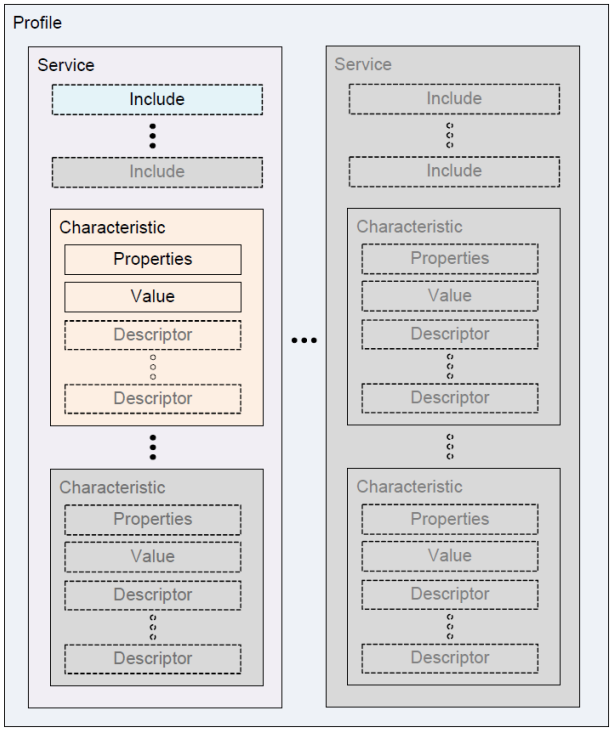
A Generic Attribute Profile (GATT) egy szolgáltatás keretrendszert definiál, amely az Attribute Protocol-t alkalmazva lehetővé teszi az egyes szolgáltatások (*Services*) felderítését, illetve az ezekhez a szolgáltatásokhoz tartozó adatok/tulajdonságok (*Characteristics*) olvasását és írását egy távoli eszközön (*Peer*). Alapvetően arra találták ki, hogy egy olyan alkalmazást vagy egy további profilt építhessünk rá, melyben kliens és szerver közti kommunikációt valósítunk meg. A 9. ábra mutatja azokat a stack-ben érintett rétegeket, amelyek szükségesek a GATT működéséhez. [2]

A GATT-ot implementáló eszközök számára a következő szerepeket definiálja a profil: [2]

- **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 és válaszol rájuk, továbbá értesítéseket és jelzéseket is küld a 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 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 oldalon található (10.) ábra szemlélteti. [2]



10. ábra [2]

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]

Included Service:

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

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]

A GATT által nyújtott legfontosabb szolgáltatások:

- Konfiguráció közlése
- *Service*-ok és *Characteristic*-ek felderítése egy eszközön
- *Characteristic* értékek írása és olvasása
- *Characteristic* értékek változásának jelzése, vagy értesítés (jelzés visszacsatolással)

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

A GAP 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: [2]

- Broadcaster
- Observer
- Peripheral
- Central

Amennyiben a Link Layer Controller támogatja, úgy egy eszköz párhuzamosan több szerepet is betölthet, az egyes Link Layer-beli tiltott állapotkombinációk betartásával. [2]

Broadcaster

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

Link Layer-beli állapot: **Advertising**

Advertising állapotban a Link Layer advertising PDU-kat küld a hirdetési csatornákon, bizonyos advertising event-ekben. Az Advertising event fajtája jelöli ki az esetleges választlehetőségeket és az ott használt PDU-kat (11. ábra). [2]

Advertising Event Type	PDU used in this advertising event type	Allowable response PDUs for advertising event	
		SCAN_REQ	CONNECT_REQ
Connectable Undirected Event	ADV_IND	YES	YES
Connectable Directed Event	ADV_DIRECT_IND	NO	YES*
Non-connectable Undirected Event	ADV_NONCONN_IND	NO	NO
Scannable Undirected Event	ADV_SCAN_IND	YES	NO

11. ábra [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**

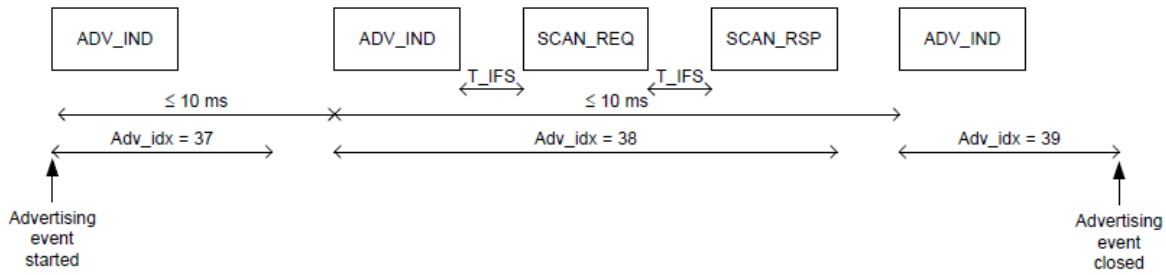
Az itt használt PDU-k formátumait összefoglaló táblázat:

PDU	Advertising Channel PDU Payload	
SCAN_REQ	ScanA (6 oktett)	AdvA (6 oktett)
SCAN_RSP	AdvA (6 oktett)	ScanRspData (0 - 31 oktett)

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]

Minden olyan esetben, amikor a Scanner elfog egy bizonyos hirdetményt, azt Advertising Report formájában jeleznie kell a Host-nak. Amennyiben duplikátumot érzékel, úgy azokról nem kell jelentést küldenie. A duplikátum ez esetben több, azonos címről érkezett hirdetményt jelöl. Ez a szűrés természetesen kikapcsolható. [2]

Scanning állapotban kétféle scanning típust határoz meg a specifikáció: Passive és Active. Passive Scanning esetén a Link Layer csak fogadja a hirdetményeket, ám azokra nem válaszol. Active Scanning esetén a Link Layer fogadja az üzeneteket és ADV_IND vagy ADV_SCAN_IND PDU-k esetén SCAN_REQ PDU-t küld a forrásnak, amelyre az SCAN_RSP PDU-val válaszol (12. ábra). [2]



12. ábra [2]

Peripheral

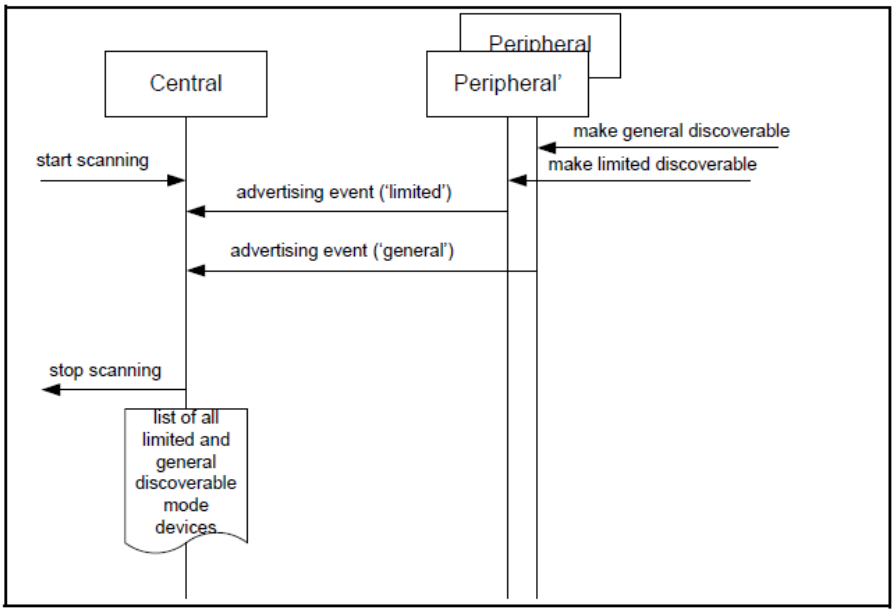
Bármely olyan eszköz, amely elfogadja egy LE fizikai kapcsolat létrehozását Peripheral szerepet tölt be. Egy ilyen eszköz a Link Layer **Connection** állapotában **Slave** szerepet fog betölteni. Ez a szerep több állapotot is felölel a Link Layerben, ugyanis **Advertising** állapotból indul, és egy CONNECT_REQ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

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 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) igénybevett, a felderítés itt igen egyszerű (akár 200-300ms is elég). Példaképpen egy általános Bluetooth LE eszköz-felderítési és kapcsolódási folyamat (13. ábra). [2]



13. ábra [2]

Két egymást követő hallgatózó ablak között 11,25ms időnek kell lennie, amely érték megegyezik magával a hallgatózási ablak hosszával, azaz a hallgatózásnak folytonosnak kell lennie. Az egész procedúra a specifikácó által ajánlottan 10,24s ideig kell, hogy tartson. Azonban a szükséges idő nagyban függ a felderítendő eszközök hirdetéseinek gyakoriságától.

Ha egy eszközt felderíthetővé szeretnénk tenni, úgy limited (rövid ideig) vagy general (hosszabb ideig) discoverable módba kell konfigurálni. A két mód között pusztán teljesítményminimalizálási célból az a fő különbség, hogy az elsőben a hirdetések közötti időközök 250-500ms között, míg a másodikban 1,28-2,56s között kell, hogy legyenek. [2]

Az első ütemben az eszköz (Central) a hirdetési csatornákon figyeli az egyes ADV_IND és ADV_DIRECT_IND PDU-kat majd listázza azok forrásainak címeit.

Az eljárás második részében a host-nak (user) kell dönteni arról, hogy melyik eszközhöz kíván kapcsolódni. Ezt követve a kiválasztott eszköz címe beíródik a kontrollerbe és a kapcsolat így létrejön egy CONNECT_REQ PDU segítségével. A kommunikáció eszt követve az adatcsatornákon zajlik. [2]

Az 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]

Elérhetőség a mobil készülékekben

A Bluetooth LE (Low Energy) technológia a legfrissebben (2010. június 30.) kiadott Bluetooth verzió (4.0) részét képezi, melyet több ízben, 2011-ben és 2012-ben is kiegészítettek, ill. javítottak. Elterjedtsége éppen ezért még igen korlátozott, azonban egyre több mobil készülékben (iPhone 4S, iPhone 5, iPad 3, iPad mini, Motorola RAZR, HTC One X stb.) jelenik meg, köszönhetően a korábbi Bluetooth rádióinterfészekbe való egyszerű implementálhatóságnak. A következő pontokban az egyes mobil platformok publikus Bluetooth LE API-jainak képességét foglalom röviden össze.

Apple iOS 5 (Core Bluetooth Framework Reference)

Az itt használt főbb függvényosztályok és ezek funkciói: [3]

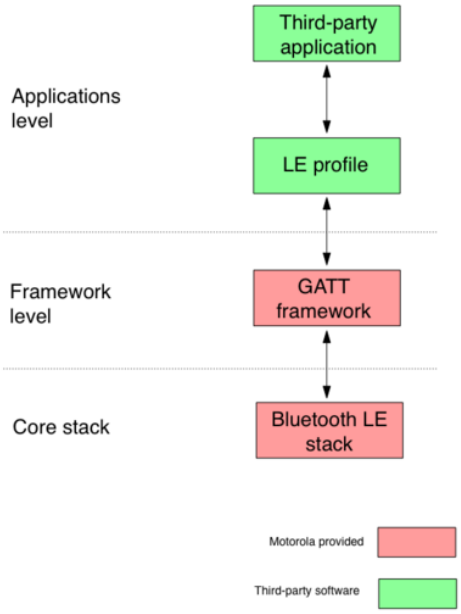
- CentralManager (discovery (scan), connect)
- Peripheral (service- and char.discovery, RSSI, read and write characteristics)
- Characteristic (value, UUID, service, properties)
- Service (UUID, peripheral, included services)

Ez az operációs rendszer rendelkezik jelenleg a legszélesebb támogatással.

Google Android 4.0 – Ice Cream Sandwhich

Ezen operációs rendszer nem fogalmaz meg támogatást. Az API-ban csak a publikus függvényekre támaszkodhatunk, hiába a BlueZ Bluetooth Stack (ami már támogatja) és hiába csalogatjuk elő az itt elrejtett függvényeket reflektálással, ha azokat a későbbi kommersz eszközökön akarjuk használni, ugyanis a reflektálás root jogot igényel az Android OS-en.

Azonban a Motorola fejlesztői kiadtak egy API-t (Motorola Bluetooth Low Energy GATT Framework API) aminek segítségével, ugyan csak Motorola telefonokon, de Android alapon fejleszthetünk Bluetooth LE alkalmazásokat. Ezen API szerveződését láthatjuk a jobb oldali ábrán. [3]



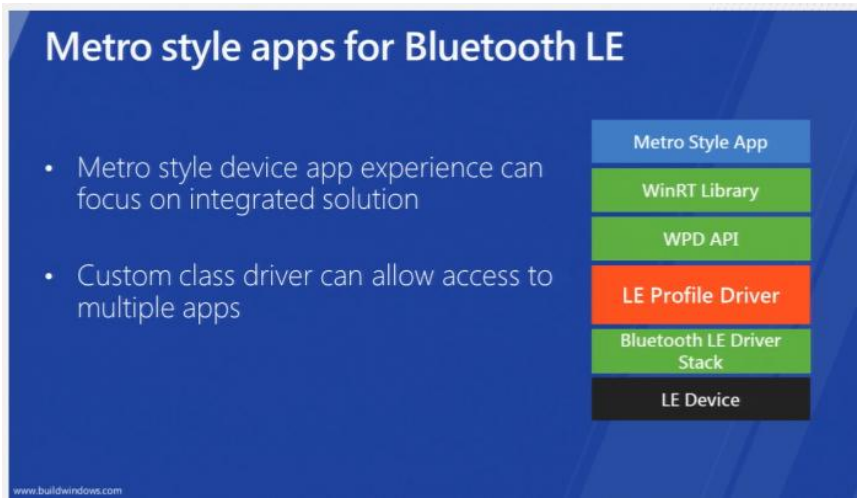
Az ezen API által nyújtott főbb funkciók:

- Connect, Disconnect
- Get primary Services
- Read, Write, Get Characteristics
- Write Characteristic Descriptor (Properties)
- Callback for Indications and Notifications

Kikötik azonban a Motorola fejlesztői, hogy az eszközök felderítését illetve a párosítást az eredeti Android Bluetooth API függvényeivel kell végrehajtani. Amennyiben ez így van, akkor némi BT LE támogatás azért belekerült az új Android-ba is, hiszen az eszközfelfelderítési procedúra gyökeresen eltér a legacy BT-tól, [3]

Microsoft Windows 8 (RT)

Idén várható ezen operációs rendszer megjelenése, amely az ígéretek szerint, mind asztali, mind mobil környezetben támogatni fogja a Bluetooth LE technológiát. [5]



Windows Bluetooth LE client API: [5]

- Get Data (Service, Characteristics)
- Set Data
- Receive Data events (Notifications, Indications)

Látható tehát, hogy ezek operációs rendszerek, csak az adatcsatornákon tudnak kommunikálni a szerverrel. Tehát egytől-egyig GATT kliens szerepében látják az operációs rendszerek fejlesztői a mobil készülékeket, mindezen felül GAP-beli Centralként. Amennyiben olyan többugrásos Bluetooth LE hálózatot tervezünk, amelyben egy mobil készülék részt vesz, úgy ezt szemelőtt kell tartani.

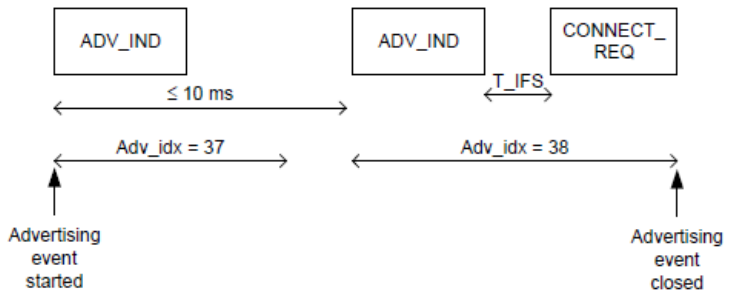
Átjárás a Piconetek között

Az átjárás kulcsfontosságú olyan esetekben, ahol nagyobb kiterjedésű, együttműködő szenzor vagy távirányítási hálózatokat szeretnénk alkalmazni. Ilyen lehet például egy nagyobb épület (irodaházak, egyetemek, raktárak) felügyeleti, riasztó és távirányítási rendszere. A megvalósítása Bluetooth Low Energy rendszerek esetén alapvetően kétféleképpen lehetséges. A master és slave szerepek gyors váltogatásával, vagy a hirdetési csatornákon történő kommunikációval. [6]

A master és slave szerepek váltogatása

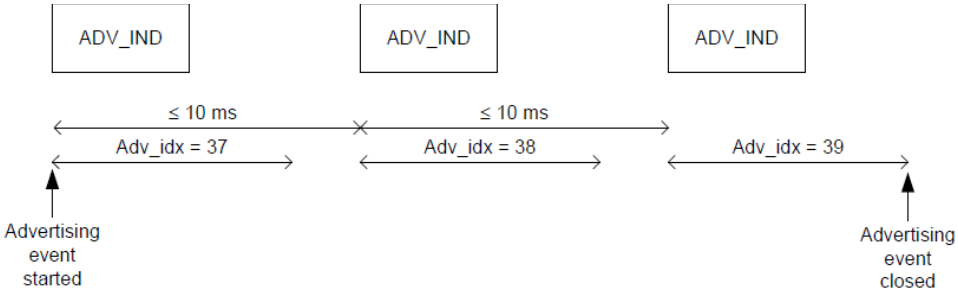
Az említett módszer lényege, hogy egy peripheral eszköz, miután megkapta a central node-tól az átviendő üzenetet kilép a piconetből, azaz terminálja a kapcsolatot és central-ként inicializál egy kapcsolatot egy másik eszközzel és átadja neki az üzenetet. Miután a csomag továbbítása megtörtént az auto-connection üzemmódot kihasználva visszaléphet a piconetbe. A folyamat iteratív módon addig ismételhető, amíg az információ el nem jut a megfelelő node-ig. Természetesen ezen módszer megvalósításához szükség van egy megfelelő ad-hoc útválasztó eljárásra, aminek segítségével az útvonalak meghatározhatók. [6]

Az ilyen állapotváltások viszonylag gyorsan ismétlődhetnek (egy eszköz indulási idejétől függően), azonban erre vonatkozóan nincsenek publikus mérések még. Saját tapasztalataim alapján egy eszköz indulása ms-os nagyságrendben történik. A szűk keresztmetszet ezen megoldás esetén mindenképpen az eszközök felderítések és kapcsolódás hosszúsága (14. ábra). [2]



14. ábra [2]

A hirdetési időközök (advInterval ≥ 20ms) figyelembevételével legjobb esetben is ms-os nagyságrendben lehetséges csak az eszközök érzékelése. Az azokhoz való csatlakozás pedig szintén ugyanennyi időbe telik. Az alábbi ábrán látható, hogy amennyiben érzékeltünk egy hirdetményt, úgy 150us (T_IFS - Inter Frame Space) elteltével válaszolhatunk csak CONNECT_REQ PDU formájában, a kapcsolat pedig ezt követve minimum 1,25ms elteltével használható. Azonban a három hirdetési csatornán véletlenszerűen adhatnak a hirdetők, így meglehet, hogy 30ms idő elteltével érzékeljünk csak az eszközt, tekintettel arra, hogy egy Advertising eventben kevesebb, mint 10 ms ideig ad a hirdető egy csatornán (15. ábra).



15. ábra [2]

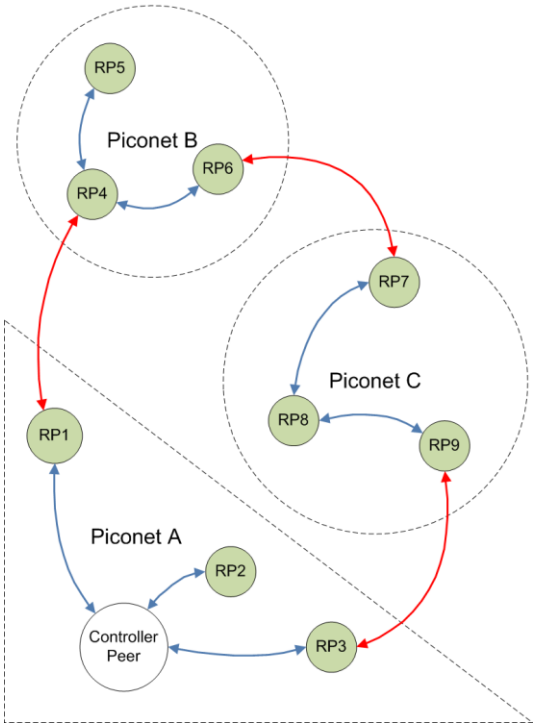
A visszakapcsolódás a piconetbe már több időbe is beletelhet, ugyanis miután az adat átadása megtörtént a GAP auto-connection eljárásával a visszakapcsolódás gyakorlatilag egy egyszerű felderítési és kapcsolódási folyamatot jelent a central eszköz részéről, annyi különbséggel, hogy a cím már ismert és a felhasználónak nem kell azt kiválasztania. Specifikációban leírtak szerint a scanWindow (amennyi ideig hallgatózunk) csak addig lehet nyitva, amíg annak fenntartása időzítési vagy ütemezési problémába nem ütközik. Ilyen probléma lehet a már meglévő kapcsolatok fenntartásához szükséges üres kontrollcsomagok küldése a piconet szétesésének elkerülése végett vagy bármilyen az adatcsatornákon történő kommunikáció, amelynek részleteire a következő pontokban térek ki. [2]

Komunikáció a hirdetési csatornákon

A fizikai kapcsolati (Link Layer) réteg több állapotgép egyidejű működését is támogathatja, az ezekre vonatkozó megkötéseket a specifikáció a következőképpen fogalmazza meg: [2]

- *Connection állapotban a Link Layer nem töltheti be egyszerre a Master és a Slave szerepét.*
- *Connection állapotban és Slave szerepben az említett rétegnek csak egyetlen kapcsolata lehet.*
- *Connection állapotban és Master szerepben lehet több kapcsolata is.*
- *A fizikai kapcsolati réteg nem működhet Initiating állapotban, ha az Slave szerepet tölt már be a Connection állapotban.*
- *Amennyiben a Link Layer Connection vagy Initiating állapotban van, akkor Advertising állapotban nem lehet olyan típusú hirdetménye, amelynek eredményeképpen az említett réteg Slave szerepben lép a Connection állapotba.*

Ahhoz, hogy az átjárás a hirdetési csatornákon megvalósulhasson egyes kitüntetett eszközöknek több Generic Access Profile-béli szerepet kell egyidejűleg betöltenie. Amennyiben az RP1-RP4, az RP3-RP9 és az RP6-RP7 nodepárok egyidejűleg töltenek be Broadcaster és Observer szerepeket (azaz a Link Layerben Advertiser és Scanner) a Piconet-beli szerepüktől függetlenül, úgy ez az átjárás megvalósítható. Ha ezek a nodepárok csak scannable undirected és non-connectable undirected eventeket alkalmaznak a kommunikáció megvalósítására, úgy a Link Layer-beli állapotaik semmilyen módon nem ütköznek a specifikáció előírásaival. Így elméletben a megoldás működőképes. [2]



16. ábra

Az ábrán (17.) három Piconet-et számlálhatunk meg. A Controller Peer az a bizonyos node amely a kommunikációt a távoli node-dal kezdeményezi. Az RP rövidítés pedig a Remote Peer kifejezést takarja és a távoli node-ot jelöli. A kék nyilak a Link Layer-beli Connection állapotra utalnak, míg a pirosak szolgálják az átjárást az egyes Piconet-ek között. [2]

Röviden leírva a működést, ha az RP1 egy non-connactable undirected eventnek megfelelő ADV_NONCONN_IND PDU AdvData mezejébe ágyazza bele, Service Data AD típus formájában az átküldendő adatszegmenst, akkor az RP4 akár passive-scanning során is képes ezt az adatot fogadni. Ugyanez vice versa ugyanígy megtörténhet. Az adatok szegmentálását és sorszámozását pedig, egyszerűen meg lehet oldani az alkalmazási szinten, mivel alapvetően kis terjedelmű adatokról, kódszavakról beszélünk.

PDU	**Advertising Channel PDU Payload**	
ADV_IND	AdvA (6 oktett)	AdvData (0 – 31 oktett)
ADV_DIRECT_IND	AdvA (6 oktett)	InitA (6 oktett)
ADV_NONCONN_IND	AdvA (6 oktett)	AdvData (0 – 31 oktett)
ADV_SCAN_IND	AdvA (6 oktett)	AdvData (0 – 31 oktett)

A statikus vagy lassan változó adatokat scannable undirected event segítségével egy SCAN_REQ PDU-t követően, SCAN_RSP PDU-val küldhetjük, a dinamikus és gyorsan változó adatokat pedig non-connectable undirected eventek segítségével. A Piconeteken belül a kommunikáció már az adatcsatornákon zajlik, a GAP-GATT-ATT-L2CAP profilok és protokollok illetve a Link Layer segítségével. [2]

A módszer hatékonysága azonban az előbbi pontban taglalt jelenségek, azaz a megfelelően hosszú és jól ütemezett hirdetési és a hallgatózási időszakok miatt hasonlóan nagy körültekintést igényel a tervezéskor. A módszer előnye az előzővel szemben, hogy nem szükséges lezárni a kapcsolatot a central eszközzel, hanem elég egy hirdetési, ill. egy hallgatózási időszakot indítani. Azonban egyértelmű hátránya a csomagok korlátozott mérete (31 oktett), így az IPv6-os csomagok nagyméretű fejlécei (40 oktett) miatt, csak szegmentáltan küldhetők át az ilyen adatok. Ideális esetben azaz, ha az első PDU-t észleli a scanner - a késleltetés minimális a hirdetési csatornák alkalmazásakor illetőleg, ha az egyes csomagok változnak a csatornák szerint, akkor egy kisebb (22-53 oktett) méretű payloaddal ellátott IPv6-os csomag is átküldhető minimális (<30ms) késleltetéssel, a hirdetések jellegéből adódóan. Ezen felül két, ugyanabban a piconetben levő slave eszköz is kommunikálhat így egymással. [2]

A valós kísérlet

Kísérletem tárgya a minimális késleltetéssel kecsegtető, hirdetési csatornákon alapuló megoldás megvalósítása volt. A megvalósításhoz két darab a Texas Instruments által gyártott CC2540 alapú mini fejlesztő csomagot (CC2540MINI-DK) használtam (18. ábra), amelyben egy USB interfésszel ellátott, virtuális soros porton keresztül hozzáférhető modul és egy kulcstartószerű, gombelemmel táplálható kisebb modul (keyfob) található. Ez utóbbi modul két fizikai nyomógombbal, gyorsulás és hőmérsékletszenzorral, ezen felül 2 LED-del van ellátva. A kiválasztáskor a több szerepben való „egyidejű" működés volt a fő szempont.

17. ábra [7]

A cc2540-es lapkák egy intel 8051-es mikrovezérlő egységgel vannak ellátva, amely az IAR Embedded Workbench fejlesztőkörnyezetben szabadon fejleszthető. Természetesen vannak más alternatívák is, azonban a Texas Instruments a Bluetooth LE stack alsóbb, fejlesztők által nem hozzáférhető rétegeit úgy valósította meg, hogy a rendszer programozása csakis az általa közzétett, bemutató jellegű kisebb alkalmazásainak módosításával valósítható meg. A fejlesztést így a különböző demóalkalmazások „reverse engineering" módszerrel történő visszafejtésével kezdtem. [7]

A mikrokontrolleren futó program alapvetően az esemény-alapú paradigmát (event-driven programming - EDP) követi, azaz az egyes BLE stack rétegeinek részben, vagy teljes egészében megfelelő programsorokat egy azok által értelmezhető eseménnyel vagy üzenettel lehetett időzítetten vagy „egyből" meghívni. A fejlesztés során így ügyelni kellett arra, hogy a sok komponenset érintő feladatok semmiképpen ne akasszák meg a végrehajtási ciklust, késleltetve a pontos időzítést megkívánó alsóbb réteket. [7]

Az általam tervezett alkalmazásban két egymástól elkülönülő piconet két peripheral (keyfob) eszköze hívatott kommunikálni egymással egy igen egyszerű séma szerint. Az egyik gombnyomásra hallgatózni kezdett, a másik pedig gombnyomásra hirdetni kezdett. Az egyes hirdetési és hallgatózási időközöket ennek megfelelően állítottam be, azaz a gomb lenyomása után első megközelítésben a hallgatózó eszköz 4 másodpercig (scanWindow) figyelte ugyanazt a hirdetési csatornát, a hirdető pedig 250ms hirdetett azokon a gomb lenyomását követően. Mindezt természetesen azt követve, hogy a két említett eszköz belépett a számára kijelölt piconetbe. A megvalósítást nehezítette, hogy nem lehetett egyértelműen megállapítani, hogy a hallgatózás melyik csatornán fog történni, így mindegyik csatornán ugyanazt az adatot kellett elküldeni.

Miután a hirdető eszköz végzett a hirdetéssel automatikusan hallgatózni kezdett szintén 4 másodpercig egy hirdetési csatornán. A hallgatózó eszköz pedig miután érzékelte a beérkezett hirdetést automatikusan hirdetni kezdett (szintén 250ms ideig). Connected állapotban a hirdetési csatornákon az események legkevesebb 100ms-os időközökkel hirdethetnek, azaz miután <30ms alatt végzett egy eseménnyel több, mint 70ms ideig nem hirdethetett. Ezen felül a hirdetési események kezdete közötti időintervallum egy véletlenszerű advDelay (0-10ms) értékkel nőhet. Mindezek alapján „worst-case” esetben (110+110+30) = 250ms szükséges ahhoz, hogy három hirdetési csomag mindenképpen átjusson (egy csatornán hallgatózik az eszköz).

Már itt látható volt, hogy amennyiben nyugtázott üzenetküldést szeretnék megvalósítani a késleltetés mindenképpen nőni fog, ugyanis nem lehet egyértelműen megállapítani, hogy hányadik hirdetést fogja a hallgatózó eszköz elkapni, így a válasszal mindenképpen várni kell, nehogy elvétse azt.

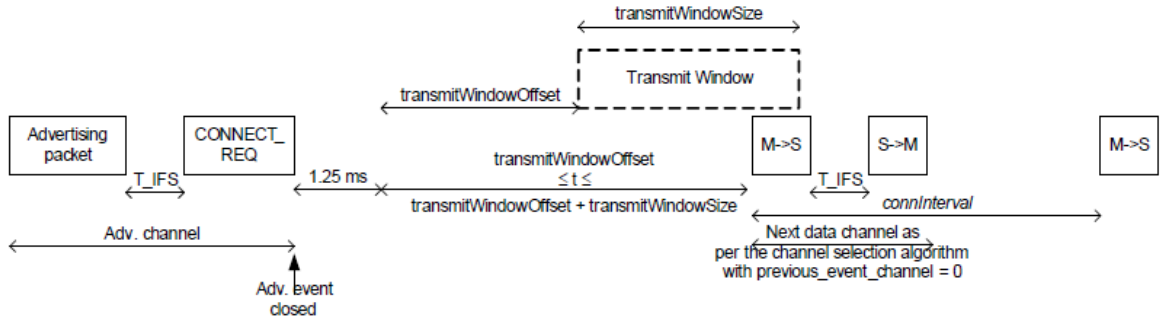
Annak észlelését, hogy egy eszköz elfogott egy hirdetményt a GATT keretrendszer értesítési funkcióját alkalmaztam. A funkció lényege, hogy, ha a GATT szerver, azaz jelen esetben a peripheral eszköz egy definiált characteristics-ben változást észlel, akkor arról nyomban értesítést küld a GATT kliensnek, azaz ebben az esetben a central node-nak. A central node-okat az USB csatlakozóval ellátott modulok testesítették meg. A TI által megírt Btool programon keresztül HCI (Host Controller Interface) üzenetekkel lehetett a modulokat vezérelni és a kívánt funkciókat aktiválni. Illetve az egyes értesítéseket látni.

Végeredményben, a hirdetés kezdetekor az egyik node értesítést küldött, egy PDU beérkezésekor a másik küldött értesítést, a nyugtázáskor pedig újra az első küldött értesítést. Az időbélyegek olvasásával pedig lehetőség nyílt az eltelt idő mérésére.

Mivel a program futtatása nem hozta meg a kívánt eredményt, vizsgálatomat a hibák lehetséges okainak feltárására fordítottam. Különösen nehezítette a feladatot az, hogy a TI által implementált Bluetooth LE stackben csak egy adott szintig (GAP) volt hozzáférés, így lehetetlen volt kideríteni, hogy a hallgatózási ablak időzítési problémák esetén való bezárása után mi történik. Az egyes kombinált szerepekben (Peripheral + Broadcaster + Observer) való működés sem volt egyértelmű, ugyanis a megfelelő paraméter átadásakor a vizsgálataim azt mutatták ki, hogy a (Peripheral + Broadcaster) kombinációhoz nem szükséges újrainicializálni az alsóbb rétegeket, hanem elég csak a hirdetést újra bekapcsolni, míg az Observer szerephez már újra kellett konfigurálni azokat. Mindez azt sejteti, hogy a belső ütemezés és a prioritáskezelés a szerepek konfigurációjától függ, és nem feltétlenül részesülnek azonos elbírálásban az azokhoz tartozó rutinok, ami az aktivitás kezdetét erősen befolyásolja. Ez azonban csak feltevés.

Így azt kezdtem vizsgálni, hogy az adatcsatornákon történő kommunikációt befolyásoló tényezőket, hogyan kell változtatni ahhoz, hogy a kívánt működés elérhető legyen.

A csatorna kiépülését az alábbi ábra (19.) mutatja.



18. ábra [2]

Egy CONNECT_REQ PDU fogadása után a transmitWindowOffset és a transmitWindowSize időkkel szinkronizálja a central a peripheral eszközt. A szinkronizáció után a kommunikáció a következő paraméterek szerint történik: [2]

- **connInterval** - Két connection event kezdete közötti idő (7,5ms – 4s)
- **connSlaveLatency** - A slave mennyi connection eventet hagyhat ki (supervisionTimeout nélkül)
- **supervisionTimeout** - Amennyiben ennyi ideig nem érkezik érvényes adatcsomag a kapcsolat szétesik (100ms - 32s).

Mindezek fényében igyekeztem úgy megválasztani a kapcsolati paramétereket, hogy a hallgatózás indulásakor a slave eszköz kihagyhasson elegendő számú connection eventet (connInterval = 80ms, connSlaveLatency = 10). Azonban ezen paraméterek beállítása esetén sem működött megfelelően a megoldás. A hibakeresés során kiderült, hogy a specifikáltakkal ellentétben a hallgatózás során az eszköz, ha kapcsolatban van, nem feltétlenül küld jelentést a beérkezett PDU-ról. Ennek oka jelenleg ismeretlen.

Ha a megfelelően hosszú hallgatózás meg is valósulhatott volna a mért késleltetés mindenképen nagy lett volna, hiszen nem tudni, hogy az éppen kimaradó connection event melyik szakaszába érkezik vissza az eszköz a hallgatózás után. Ez probléma megoldható a minimális (7,5ms-os) connInterval értékekkel, azonban a connSlaveLatncy nem lehet nagyobb 499-nél. Így az elérhető maximális kimaradási idő 3,742s. Az aktív adatcsatornákon történő kommunikációba való visszatérés ideje pedig szintén minimalizálható ezekkel az értékekkel. [2]

Ugyanakkor éppen a connSlaveLatency miatt, amíg hirdet az eszköz, addig semmilyen, a master felől érkező adatcsomagra nem tudna reagálni. Ez central felé mutatott válaszidőt növeli, ami az adatfolyam másik oldalán (central peripheral) okoz késleltetést. A scanWindow értékek alacsonyan tartásával ez a válaszidő lecsökkenhet, azonban a hirdetések elvétésének valószínűsége ezzel együtt megnőhet. A mindkét oldalon elérhető késleltetés-minimumhoz szükséges paraméterek kiszámításához mélyebb elemzés szükséges.

Annak meghatározása, hogy lehetséges-e a hirdetmények adatait olyan sebességgel változtatni, ami elegendő ahhoz, hogy az egyes hirdetési csatornákon más és más adatok jelenjenek meg nem volt lehetőség. A TI (Texas Instruments) által biztosított Packet Sniffer alkalmazás nem volt képes az összes hirdetmény érzékelésére, így ez a lehetőség is kérdéses. Ugyanakkor, ha lehetséges is lenne a hallgatózó eszköznek gyakorlatilag teljes szinkronban kellene váltani a hirdetési csatornák között, aminek a valószínűsége igen csekély. Így a piconetek között így elérhető adatsebesség korlátozódik a lehetséges 31 oktett/advertising event értékre, ami maximum 279 oktett/s-ot jelent, és az IPv6 csomagok átvitele így csak két részletben történhet, legalább 140ms-s késleltetéssel. [2]

További kérdés, hogy mennyi idő elteltével jelenik meg az első hirdetmény, amennyiben az eszköz kapcsolatban van. Ennek méréséhez komolyabb műszerek szükségesek. Amennyiben nincs kapcsolatban az eszköz, a TI által közzétett „néhány” ms-os érték a mérvadó.

Konklúzió

A leírtak alapján belátható, hogy a minimális késleltetéssel és megfelelő adatátviteli sebességgel bíró többugrásos Bluetooth LE hálózatok létrehozása nem triviális feladat. Mindkét bemutatott megoldásnak vannak előnyei és hátrányai.

A master és slave szerepek váltogatásával a nettó adatsebesség nagyobb, azonban a késleltetéseket is beleszámolva lehetséges, hogy a hirdetési csatornákon történő adatátvitel nem csak alacsonyabb késleltetést, hanem gyorsabb átvitelt is lehetővé tesz. Azonban nincs erre vonatkozóan jelenleg releváns mérés/információ, így a további vizsgálatom tárgyát fogja képezni a módszer.

A hirdetési csatornákon történő kommunikáció első ránézésre ígéretesnek tűnt, azonban kitűnt, hogy az elméleti késleltetés a gyakorlatban csak igen nehezen érhető el. Megfelelő időzítési paraméterekkel azonban megvalósítható. A hirdetményekről történő jelentés hiányának lehetséges oka, a még nem teljesen kész BLE stack. A legutóbbi frissítés idén áprilisban jelent meg, így újabb frissítések várhatók.

Elképzelhető egy hibrid megoldás amelyben a piconetek között csak protokolláris jelzéseket viszünk át a hirdetési csatornákon, a tényleges adatokat azonban kitüntetett ügynökökön keresztül küldjük, amelyek a váltogatást végzik.

Összességében a vizsgálat több kérdést is felvetett, amelyek közül néhányat sikerült megválaszolni. A megválaszolatlanok a további vizsgálataim tárgyát képezik.

Irodalomjegyzék

[1] R. Gawera, „The revolutionary opportunities of Bluetooth low energy,” EE Times Asia, [Online]. Available: http://www.eetasia.com/ART_8800637772_590626_NT_d07902d8.HTM. [Hozzáférés dátuma: 10 október 2012].

[2] V. Samosuyev, „Bluetooth Low Energy Compared to Zigbee and Bluetooth Classic,” Mikkeli University of Applied Sciences, 2010.

[3] A. Z. R. B. M. A. O. M. Zareei, „A Comparative Study of Short Range Wireless Sensor Network on High Density Networks,” in *2011 17th Asia-Pacific Conference on Communications (APCC)*, Sutera Harbour Resort, Kota Kinabalu, Sabah, Malaysia, 2011.

4] J. Decuir, „Communications Society, Bluetooth 4.0: Low Energy,” IEEE, 2010. [Online]. Available: http://chapters.comsoc.org/vancouver/BTLER3.pdf. [Hozzáférés dátuma: 19 április 2012].

5] „Transmission of IPv6 Packets over BLUETOOTH Low Energy,” IETF, [Online]. Available: https://datatracker.ietf.org/doc/draft-ietf-6lowpan-btle/. [Hozzáférés dátuma: 12 október 2012].

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

7] „DevCenter,” Apple, [Online]. Available: https://developer.apple.com/. [Hozzáférés dátuma: 10 október 2012].

8] „MOTOROLA BLUETOOTH LOW ENERGY API,” Motorola Mobility, [Online]. Available: http://developer.motorola.com/docs/bluetooth-low-energy-api/. [Hozzáférés dátuma: 10 október 2012].

9] „Bluetooth Low Energy (LE) Generic Attribute (GATT) Profile Drivers,” Microsoft, [Online]. Available: http://code.msdn.microsoft.com/windowshardware/Bluetooth-Generic-4f4ea968. [Hozzáférés dátuma: 10 október 2012].

10] „MasterSlaveSwitch,” Texas Instruments, [Online]. Available: http://processors.wiki.ti.com/index.php/MasterSlaveSwitch. [Hozzáférés dátuma: 12 október 2012].

„CC2540 Mini Development Kit,” Texas Instruments, [Online]. Available:

11] http://www.ti.com/tool/cc2540dk-mini. [Hozzáférés dátuma: 12 október 2012].