Budapesti Műszaki és Gazdaságtudományi Egyetem

Automatizálási és Alkalmazott Informatikai Tanszék

Energiahasználat optimalizálási megoldások tervezése Android platformon

Tudományos Diák konferencia dolgozat

2013.

Szerző:
Braun Patrik János EQZT6L bra.patrik@gmail.com

Konzulens:
Ekler Péter Egyetemi adjunktus ekler.peter@aut.bme.hu
Automatizálási és Alkalmazott Informatikai Tanszék

Tartalom

1 Bevezetés

Az elmúlt években a mobilplatformok népszerűsége folyamatosan nőtt. A különböző eszközök, amelyek nem feltétlenül szükségesek egy telefonhívás lebonyolításához (Pl.: Wi-Fi, GPS) már az alacsony kategóriás készülékekbe is belekerültek. A felhasználók nemcsak telefonálásra, SMS-ezésre használják telefonkészülékeiket, hanem e-mail-ezésre, navigálásra, zene hallgatásra, internetezésre, játékra és egyéb olyan szoftverek futtatására, amelyek korábban csak PC-n voltak elérhetőek. A mára széles körben támogatottá vált Multisaking-nak köszönhetően, ezeket az alkalmazásokat sokszor párhuzamosan futtatják. Az alkalmazások futása nagyban befolyásolja a készülék energia fogyasztását. Miután a mobil készülékék a mobilitásukból adódóan nincsenek folyamatos tápellátásra kapcsolva, az akkumulátor lemerülésének ideje fontos paraméter egy mobil készüléknél. Emiatt a gyártók mellett a fejlesztőknek is fegyelmet kell fordítaniuk arra, hogy ezt az időt minél hosszabbra nyújtsák. A fejlesztők ezt, az energiahatékony algoritmusok felhasználásával tudják elérni. Az ilyen algoritmusok létrehozásához a fejlesztőknek szükséges tudniuk, hogy az egyes erőforrások, amelyeket az alkalmazásiak használnak, milyen energia igénnyel rendelkeznek, illetve milyen eszközökkel lehet ezeket az energia igényeket csökkenteni. Mindezek mellett természetesen a felhasználóknak a legnagyobb érdeke az egyes töltések közötti idő meghosszabbítása. Így felmerül az igény egy olyan tudásbázisra, ami segíti a felhasználókat abban, hogy hogyan tudják eszközeiket úgy használni, hogy ezt a töltések közötti időt a lehető leghosszabbra nyújtsák.
Munkám során ezekre a felmerülő igényekre nyújtottam megoldást. Terveztem és implementáltam egy rendszert, amely képes szoftveres úton mérni az egyes erőforrások energia igényét. Azonosítottam a készülékek nagyobb fogyasztóit, és e fogyasztók energia igényét mértem meg külön-külön és kombinálva, hogy általánosan meghatározhassam az energiaigényüket. A kapott eredmények alapján pedig olyan eszközöket és módszereket terveztem és készítettem a fejlesztők számára, amelyek segítik az energiahatékony alkalmazások készítését. Továbbá a felhasználók számára egy olyan tudásbázist alakítottam ki, amely segíti őket a készülékek energiatudatos használatában.
A dolgozatom a következőképpen épül fel:

- 2. fejezetben bemutatom a kitűzött feladatot és célt
- A 3. fejezetben ismertetem, hogy ezen a területet milyen eredmények születtek korábban
- A 4 fejezet a munkám során elkészült segéd szoftvereket mutatja be.
- Az 5 fejet bemutatja a mérések során kapott eredményeket.

- 6. fejezetben a mérési eredmények segítségével kidolgozott energiahatékony szoftverfejlesztési módszereket mutatom be.
- A 7. fejezet a mérési eredmények segítségével kidolgozott, felhasználóknak szánt energiatudatos készülék használati módokat mutat be.
- 8. fejezet foglalja össze az eredményeket és itt mutatom be a további lehetséges kutatási irányokat

2 A feladat és a kitűzött célok bemutatása

A munkám során az volt a célom, hogy a fejlesztőknek olyan eszközöket és kódmintákat adjak, amelyek elősegítik az energiahatékony algoritmusok létrehozását. A felhasználóknak pedig olyan tudásbázist szerettem volna nyújtani, amely elősegíti a készülékeik olyan módú használatát, amely meghosszítja a töltések közti időt.

Ezen cél eléréséhez megvizsgáltam, hogy Android platformon milyen módon lehet energia fogyasztást szoftveresen mérni. Terveztem és implementáltam egy olyan rendszert, ami képes a készülékek erőforrásainak energia fogyasztását mérni, külön-külön és kombinálva. Ezeket a méréseket többször, több hardveren, különböző körülmények között futtattam. Ilyen körülmények voltak például a gyenge és erős jelerősséggel rendelkező hálózathoz való csatlakozás. Erre azért volt szükség, hogy általánosabban meghatározhassam az egyes erőforrások energia igényeit és kimutassam, hogy az egyes erőforrások hasonló energia igénnyel rendelkeznek a különböző hardvereken. A kapott eredmények alapján pedig olyan kódmintákat, eszközöket és módszereket terveztem és készítettem a fejlesztők számára, amelyek segítik az energiahatékony alkalmazások készítését.

Továbbá az eredmények alapján a felhasználók számára is egy olyan tudásbázist alakítottam ki, mely segíti őket a készülékek energiatudatos használatában, ezáltal lényegesen megnövelve a töltési ciklusok közötti időt.

3 Irodalomkutatás

A mobileszközök energiafogyasztásával többen is foglalkoztak. A 2009-es Google IO konferencián bemutatott „Coding for Life—Battery Life, That Is”[2] dokumentum, ehhez a dolgozathoz hasonló módon mérési eredményeket prezentál és azok alapján nyújt energiahatékony fejlesztést elősegítő eszközöket. A „Survey on Energy Consumption Entities on the Smartphone Platform”[3] munkában a szerzők több erőforrásra kiterjedő energiaigény mérések eredményeit mutatják be. Az „Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications”[4] című munka az adattovábbításra összpontosít. Mérések segítségével megvizsgálták az adatküldést és fogadást több különböző technológián (Wi-Fi, 3G) és egy *TailEnder* nevű protokollt alkottak meg, amely segíti csökkenteni az adatátvitellel fellépő energiaigényt. Egy további publikáció, melyben BME-s részvétel is volt a „Modeling resource constrained BitTorrent proxies for energy efficient mobile content sharing”[5]. A dokumentum a BitTorrent tartalom átvitelre ad egy energiahatékonyabb megoldást, úgy, hogy nem direkt a BitTorrent tölti le az adatot, hanem létrehoznak egy proxy-t, ami nagyobb sebességgel küldi az adatot, így csökkentve az energiafogyasztást. A „Distributed BitTorrent Proxy for Energy Efficient Mobile Content Sharing”[6] munka az előző publikációra épül. A BitTorrent hatékonyságát több proxy beiktatásával növelték. Szintén BitTorrent energiahatékonyságával foglalkozó dokumentum a „CloudTorrent – Energy-Efficient BitTorrent Content Sharing for Mobile Devices via Cloud Services”[7]. A BitTorrent alapú fájlmegosztás energiahatékonyságát úgy növeli, hogy a fájlokat BitTorrenten keresztül egy szerver tölti le, és az küldi egy energiahatékonyabb protokollon keresztül tovább a mobil eszköznek. A Nokia Research Center munkája, az „Energy efficiency of mobile clients in cloud computing”[9] dokumentum, amelyben a mobil eszközök a cloud computing-al kapcsolatos energiaigényét vizsgálják.

A „SleepServer: A Software-Only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments”[8] munkában a szerzők nem mobil, hanem PC-s környezetben szoftveres megközelítéssel csökkentik az eszközök energiaigényét. Létrehoztak egy SleepServer architektúrát és annak implementációját, mely lehetővé teszi, hogy a host-ok alacsony fogyasztású alvómódba legyenek képesek kapcsolni, ha nincs rájuk szükség. Így 60-80% energiamegtarkítást is el tudtak érni.

Az előző kutatások főként egyedi energia fogyasztási esetekre koncentráltak, míg én a munkámban azt tűztem ki célul, hogy megvizsgáljam és megmérjem az energiafogyasztás

mértékét a gyakran használt alkalmazási módoknak fejlesztési és felhasználási szempontból és mintákat, valamint ajánlásokat fogalmazzak meg, melyekkel az energiahasználat csökkenthető.

4 A vizsgálathoz szükséges mérőrendszer tervezése

A mérések elvégzéséhez elsőként egy Android szoftvert terveztem és valósítottam meg, amely egyben monitorozza a telefon állapotát, valamint teszteket is tartalmaz, amelyek az egyes erőforrásokat terhelik. Így a monitornaplóból megállapítható, hogy az egyes tesztek során mennyi volt a készülék fogyasztása és megismerhető az egyes erőforrások energiaszükséglete is.

Az Android alkalmazás mellett szükség volt még további segéd programokra is, amelyek a tesztekhez állítottak elő bemenetet, illetve a hálózati teszt szerverét valósították meg.

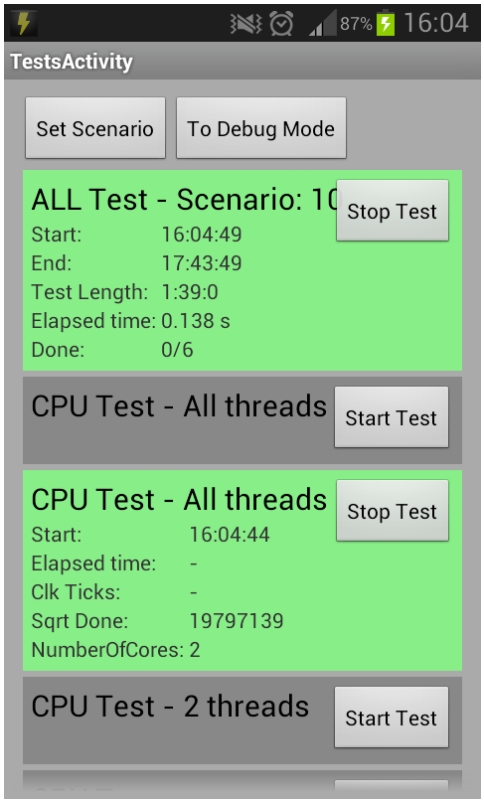
4.1 PowerMonitorAndTester

A PowerMonitorAndTester alkalmazás egy a mobil eszközön futó szoftver, amely a monitoring modult és az erőforrásokat terhelő teszteket tartalmazza.

Ebben a fejezetben egy áttekintést adok a szoftver felépítéséről és az elérhető elemi tesztekről. Az elemi tesztek szolgálnak, arra, hogy a teszt futása alatt az egyes erőforrásokat folyamatosan használják. Így az adott pillanatban mért készülék áramfelvételekéből következtetni lehet az egyes erőforrások energia igényére.

Az alkalmazást futás közben az 1. ábra és a 2. ábra mutatja be. Az 1. ábra az alkalmazás indulásakor megjelenő főképernyőt mutatja be. Itt lehet indítani a monitort. A 2. ábra a tesztek képernyőt mutatja be. Itt lehet indítani az egyes teszteket, akár párhuzamosan is.

1. ábra PowerMonitorAndTester főképernyője



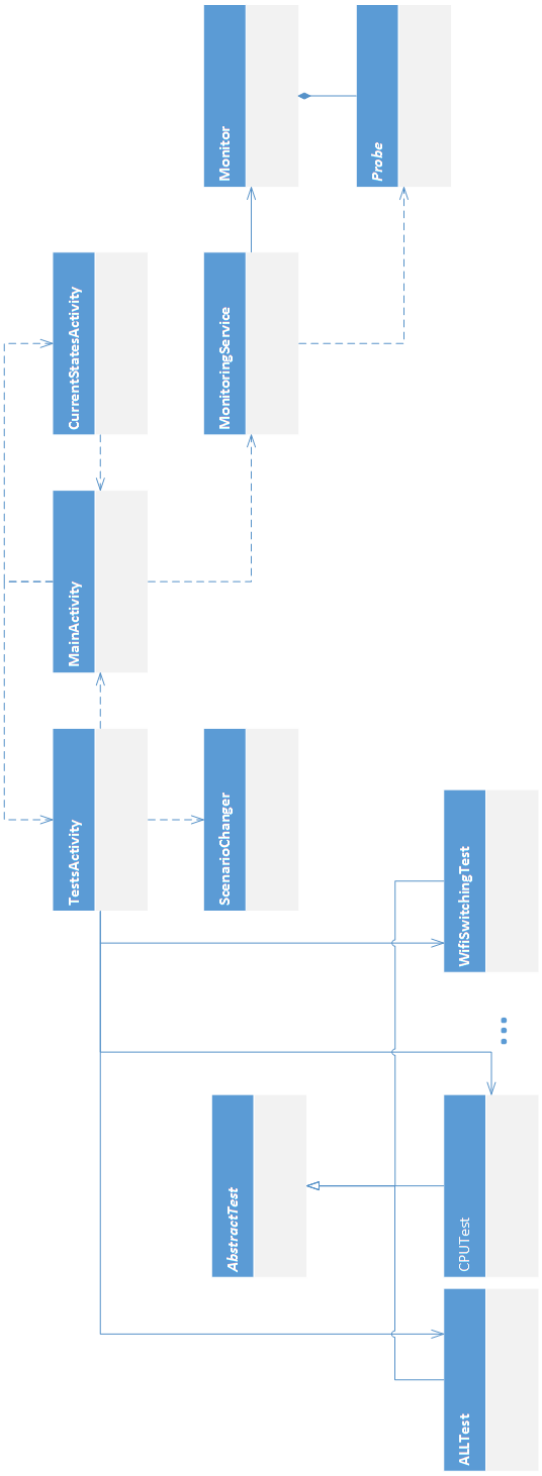
2. ábra PowerMonitorAndTester tesztek képernyője

4.1.1 Architektúra

A program architektúráját az alábbi UML diagramok mutatják be. Az ábrák az egyszerűbb érthetőség kedvéért csak a szükséges osztályokat tartalmazzák.

4.1.1.1 Áttekintés

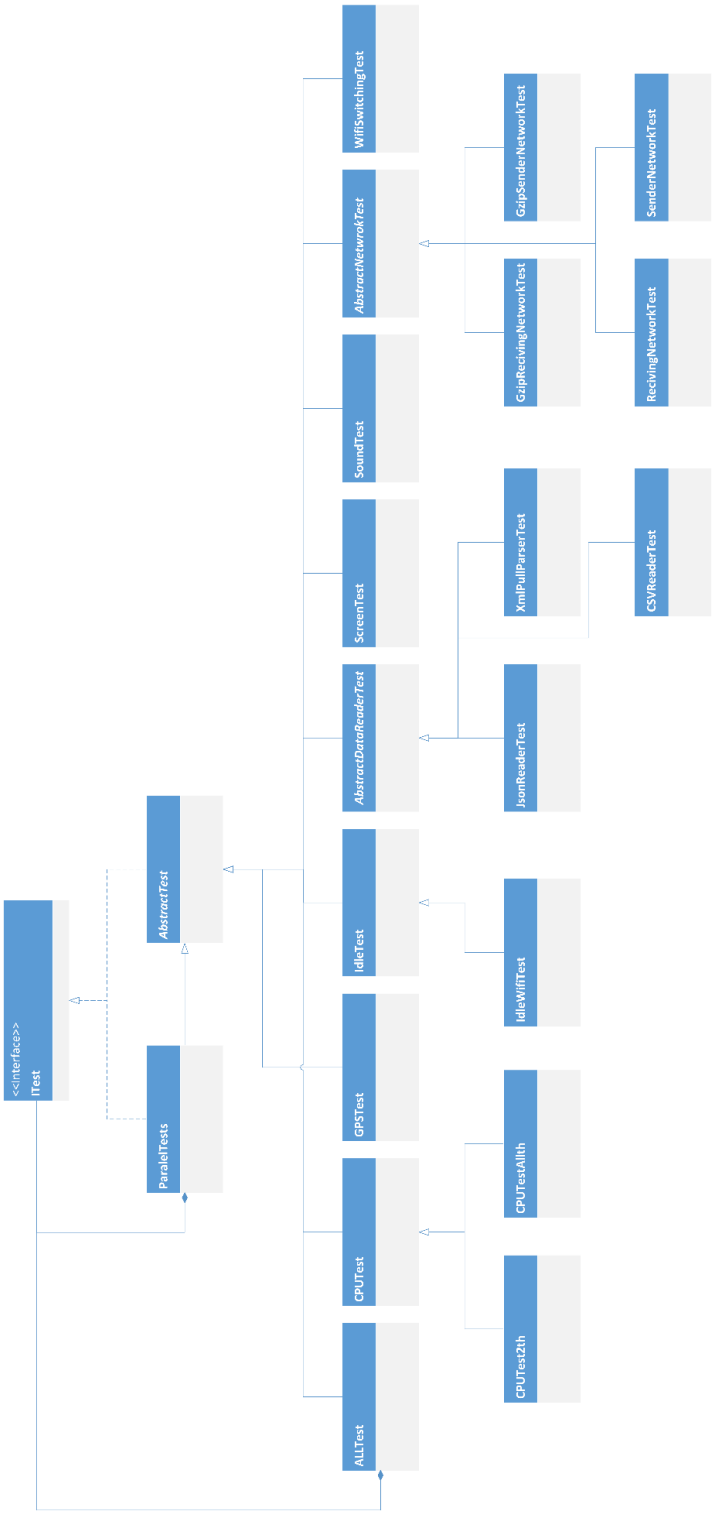
Az 3. ábra a *PowerMonitorAndTester* architektúráját ismerteti. A programban a *MainActivity* jeleníti meg a kezdő képernyőt. Itt lehet a monitorozó szervizt elindítani és átnavigálni a további képernyőkre. A felületek között található egy teszteket megjelenítő nézet, amelyen az összes elemi teszt szerepel és egy *CurrentStateActivity* képernyő, amely az utolsó monitor bejegyzést mutatja meg. A *MonitoringService* az a szerviz, ami monitorozásért felelős. Futás közben periodikusan meghívja a saját *Monitor* példányán a naplózás függvényt. A függvény a szerviz által korábban átadott *Probe* listán végig megy és a *Probe*-okból kinyert adatokat egy csv fájlba menti. A Probe egy absztrakt osztály, amelyet felül definiálva meg lehet adni, hogy a *Probe* futása során melyik erőforrásról adjon információt. Példa egy ilyen *Probe*-ra a CPU aktuális terheltsége.



3. ábra PowerMonitorAndTester architektúra áttekintése

4.1.1.2 Tesztek

A 4. ábra az elemi tesztek architektúráját mutatja be. Az *ITest* interfészt valósítja meg az összes elemi teszt. A tesztek által gyakran használt, vagy ugyanolyan módon megvalósított függvényeket az *AbstactTest* absztrakt osztályba gyűjtöttem.



4. ábra PowerMonitorAndTester tesztjeinek architektúrája

4.1.2 Monitor

A Monitorozó szerviz a 4.1.1.1 fejezetben említett *Probe*-okat tárol egy listában. Minden futáskor ezeket a *Probe*-okat futtatja. A futás eredményét pedig egy csv fájlba menti. A *Probe*-ok futtatását periodikusan végzi. A mérések során én 5 másodpercre választottam a periódus

időt. A monitor kimenetére a 1. táblázat ad egy példát. A táblázatban a jobb ábrázolhatóság végett felcseréltem az oszlopokat és sorokat. Így az első oszlop a CSV fejlécének felel meg, a többi pedig az egyes monitor futás eredményeinek.

Time	2013. 10. 5. 19:00:12	2013. 10. 5. 19:00:16	2013. 10. 5. 19:00:21
timeStamp	1380992411611	1380992415924	1380992420925
Comments	New File Started (device: golden ,mdoel: GT-I8190 version: 4.1.2)		
CPU work [ticks]	2611323	2611775	2612615
CPU idle [ticks]	8997137	8997546	8997708
Mobile Received [bytes]	0	0	0
Mobile Transmitted [bytes]	0	0	0
Mobile Received [packets]	0	0	0
Mobile Transmitted [packets]	0	0	0
All Received [bytes]	2118301068	2118301800	2118301800
All Transmitted [bytes]	637619755	637620379	637620379
All Received [packets]	2279230	2279243	2279243
All Transmitted [packets]	1179616	1179624	1179624
Wifi Rssi	-37	Radio OFF	Radio OFF
GPS enabled	false	false	false
GPS satellites[Max]	0	0	0
GPS satellites[used]	0	0	0
GPS satellites[fix]	0	0	0
GPS location	N/A	N/A	N/A
Battery Currnet [mA]	-176	-176	-220
Battery Level	88	88	88
Battery scale	100	100	100
Battery voltage [mV]	4124	4124	4087
Battery temp	346	346	350
Brightness	57	57	57
Screen	ON	ON	ON
clockTicks	3	2	2
RunTime [ms]	59	119	34

1. táblázat Monitor kimenet példa

4.1.3 Elemi tesztek

Az elemi tesztek egy jól meghatározott erőforrást használnak abból a célból, hogy a készülék pillanatnyi energiafogyasztását felhasználva, megállapítható legyen az adott erőforrás energiaigénye. Az egyes teszteknek a tesztelésen kívül más felelősségük nincsen. Így ha a teszthez szükséges hálózati kapcsolat, azt a teszt indítójának kell biztosítani.

4.1.3.1 Csoportos teszt

Ez a teszt nem tartozik szorosan az elemi tesztek közé. Azért itt kerül megemlítésre, mert architektúra szempontjából ugyanolyan, mint bármelyik elemi teszt. A csoportos teszt képes a megkapott *ITest* interfészű teszteket egymás után futtatni. A futtatás során, ha valamelyik tesztnek szüksége van valamilyen erőforrásra, mint például Wi-Fi, bekapcsolt képernyő, akkor ezeket az ezeket az erőforrásokat biztosítja. A teszteket előre megadott ideig futtatja és futtatott tesztek között szünetet tart, hogy a készülék energiafogyasztása normalizálódni tudjon.

4.1.3.2 Egy szálon futó CPU teszt

Az egy szálon futó CPU teszt a futása során, egy háttér szálon folyamatosan gyökvonásokat végez. Az algoritmust a következő *pszeudokód* mutatja be:

```
SET kezdés = 10000
SET intervallum = 10000
WHILE a teszt fut
      FOR i = kezdés TO kezdés + intervallum
            CALL gyökvonás with i
      ENDFOR
      CALL Sleep with 1
ENDWHILE
```

A gyökvonást mindig más számmal végzem, hogy a futtató környezetnek és a fordítónak ne legyen lehetősége optimalizálni a kódot. A gyökvonást azért szakítom meg időként egy *sleep* hívással, hogy a futtató környezetnek legyen lehetősége más szálakat is beütemezni.

4.1.3.3 2 szálon futó CPU teszt

A két szálon futó CPU teszt megegyezik az egy szálon futó tesztel, csak a háttérszálból, amely a gyökvonásokat végzi, két darabot indít.

4.1.3.4 Összes szálon futó CPU teszt

Az összes szálon futó CPU teszt megegyezik az egy szálon futó tesztel, csak a háttérszálból, amely a gyökvonásokat végzi, annyi darabot indít, ahány mag rendelkezésre áll az eszközön.

4.1.3.5 GPS teszt

A GPS teszt feliratkozik az *LocationManager::requestLocationUpdates* függvényére úgy, hogy a frissítések közti minimum idő és távolság 0ms és 0m. Ezzel eléri azt, hogy a készülék a műholdas helymeghatározási rendszer segítségével meghatározza a készülék szélességi és hosszúsági koordinátáit. Majd ha sikerült a meghatározás, folyamatosan értesítse a tesztet a pozíció változásokról. Így a teszt eléri, hogy a készülék használja a műholdas helymeghatározó erőforrását. A teszt a nevével ellentétben nem csak az amerikai GPS műholdakat használja helymeghatározásra, hanem, ha a készülék támogatja, akkor másokat is, például GLONASS.

4.1.3.6 Rendszer üresjárat teszt

Ez a teszt nem használ dedikált erőforrás. A rendszer üres járási energiafogyasztását segít mérni. A teszt azért került létrehozásra, hogy a rendszer üres járási energia fogyasztását is hasonló körülmények közt lehessen mérni, mint bármelyik más erőforrást. Ez az energiafogyasztás nem egyezik meg a telefon alvó módjával járó energiafogyasztásával, mert a teszt wakelock-ok segítségével megakadályozza, hogy a készülék alvó módba menjen. Az alvó mód energiafogyasztása szoftveres úton nem mérhető, mert, alvó módban nem tud futni az a monitor, ami képes kiolvasni a készülék pillanatnyi energiafogyasztását.

4.1.3.7 Készenléti Wi-Fi teszt

A teszt lényegében megegyezik 4.1.3.6 Rendszer üresjárat teszttel, annyi a különbség, hogy a teszt futtatásához szükség van bekapcsolt Wi-Fi modulra.

4.1.3.8 Fájl feldolgozás tesztek

A rendszer tesztel CSV, XML és JSON formátumú fájlok passzolását. Teszteket úgy végeztem, hogy létrehoztam a *Person* osztály a következő adattagokkal: *Name, Address, PhoneNumber, Gender, Profession, FavouriteColor, ShoeColor*. A *Person* oszályból a *DataCreator* segéd alkalmazás segítségével generáltam 50000 darabot, majd ezeket kiírtam CSV, XML illetve JSON formátumban, külön fájlokba. Az egyes fájlok adatait a 2. táblázat tartalmazza.

Formátum	Sorok száma	Fáj méret
CSV	50002	4587 KB
JSON	450002	11862 KB
XML	450003	16318 KB

2. táblázat CSV, XML, JSON formátumú fájlok adatai.

Az adatok beolvasásakor az alkalmazás direkt kihagyja a *FavouriteColor, ShoeColor* adatokat. Ezzel azt helyzetet szimuláltam, amikor egy adattal teli fájlt olvasunk be, nem minden adatra van mindig szükségünk.

Az egyes formátumú adatok feldolgozásához próbáltam mindig a legegyszerűbb, megoldást használni, hiszen a fejlesztők többsége is ezt teszi. A feldolgozáshoz a következő osztályokat használtam:

- CSV: com.csvreader.CsvReader
- JSON: android.util.JsonReader
- XML: org.xmlpull.v1.XmlPullParser

A CSV beolvasásához egy ismert algoritmus használtam[1], a JSON és XML feldolgozáshoz pedig az Android platformon megszokott és gyakran felhasznált módszert alkalmaztam.

4.1.3.9 Képernyő teszt

A képernyő teszt a képernyő energiafogyasztását segíti meghatározni különböző fényerősségek esetén. A teszt az indulásokat paraméterként megkapott tesztelési idő hosszát osztja 4 felé. Az egyes időszeletben a következő fényerőségeket állítja be:

1. 100% fényerő
2. 50% fényerő
3. 0.1% fényerő (a legkisebb beállítható fényerő)
4. kikapcsolt képernyő. Ez megegyezik a Rendszer üresjárat teszt-el

4.1.3.10 Hang teszt

A teszt a beépített hangszóró (csatlakoztatott headset esetén, a headset) energiafogyasztását segít meghatározni. A Képernyő teszthez hasonlóan ez is 4 egyenlő részre osztja a paraméterként kapott tesztelés idejét. Az egyes időszeletben pedig a következő hangerősségeket állítja be:

1. 100% hangerő
2. 50% hangerő
3. 0.01% hangerő
4. 0% hangerő

A teszteléshez Vivaldi négy évszak, tavasz I. Allegro molto-t mp3 formátumban használom.

4.1.3.11 Adatfogadás teszt

A hálózaton keresztül küldött adat fogadásának energia igényét segíti felmérni. Ezt úgy teszi meg, hogy kapcsolódik a *NetworkTesterServer* szerverhez és megkezdődik az adatfogadás. Az adatokat TCP/IP felett küldöm. Az adatok fogadásának algoritmusát a következő pszeudokód mutatja be:

```
OPEN socket
SET loopCount = 10
WHILE a teszt fut
      FOR i = 0 TO loopCount
            READ 1024 bytes FROM socket
      ENDFOR
      CALL sleep with 1
ENDWILE
```

[1] www.csvreader.com

Az algoritmus implementálásában *java.io.DataInputStream*-en keresztül küldöm az adatokat. Az algoritmusa azért több részeltben fogad, mert a küldő oldal is ilyen felosztásban küldi ki az adatot. Erre azért volt szükség, mert a megfelelő sávszélességet így tudtam elérni. Az ütemezés miatt szükségszerű volt a szálat időnként sleep-el megszakítani, adatátvitelnél pedig jobb volt legalább ekkora adatmennyiséget egyszerre kiküldeni, mert így több adatott tudott egy csomagban elküldeni, így kisebb volt a csomag fejlécek miatti overhead a hasznos adathoz viszonyítva.

4.1.3.12 Adatküldés teszt

Az adatküldés energiaigényének mérése nagyban megegyezik az adatfogadás energiaigényének mérésével. A különbség abban nyilvánul meg, hogy fogadás helyett küldés van és *java.io.DataInputStream* helyett *java.io.DataOutputStream*-t használok. Több különbözőbb adatot is próbáltam küldeni. Eleinte 0-al feltöltött byte tömböt, majd a küldés elején véletlen értékkel inicializált byte tömböt. Ezen két megoldás közül egyik sem volt megfelelő tömörített csatornán való adat küldéshez, ezért végül egy előre létrehozott txt fájl tartalmát olvasom be és küldöm át a hálózaton. A szöveg file a Star Wars részek forgatókönyvét tartalmazza.

4.1.3.13 GZIP adatfogadás teszt

A GZIP adatfogadás teszt algoritmusa megegyezik az Adatfogadás teszt algoritmusával, a különbség az annyi, hogy a *java.io.DataInputStream*-t nem közvetlenül a létrehozott *socket outputStream*-jéből hozom létre, hanem létrehozok egy *java.util.zip.GZIPInputStream*-t a kettő közé. Így az átküldött adat GZIP-el tömörítve lesz.

4.1.3.14 GZIP adatküldés tesz

Ez a teszt megegyezik a GZIP adatfogadás teszttel, csak *inputStream*-ek helyett *outputStream*-eket használok

4.1.3.15 Wi-Fi be/ki kapcsolás teszt

Ez a teszt a Wi-Fi antenna be és kikapcsolásával járó energia igényt segít meghatározni. A teszt futása a következő pszeudokóddal írható le:

```
WHILE a teszt fut
      IF Wi-Fi modul be van kapcsolva AND (hálózathoz kapcsolódva OR
timeout)
            Wi-FI modul kikapcsolása
      ELSE
            Wi-Fi modul bekapcsolása
      ENDIF
ENDWHILE
```

4.1.4 Alkalmazás tesztelése

Az alkalmazás fejlesztése során az elkészült funkciók tesztelése egy fontos lépés. Az alkalmazás méretéből adódóan nem éreztem szükségét tesztek létrehozására. Az eclipse alapú Android SDK jó debugolási lehetőségekkel rendelkezik. Az Androidba beépített LogCat segítségével, pedig az alkalmazás futása során keletkezett log-okat tudtam megtekinteni, ami nagy segítség volt a fejlesztés során. Ezen eszközök mellett még használtam ACRA-t (Application Crash Report for Android), ami az el nem kapott kivételeket automatikusan egy beállított online adatbázisba tölti fel.

4.2 Segéd Szoftverek

4.2.1 NetworkTesterServer

A program az adat küldés és fogadás tesztekhez hozza létre a szervert, ahová a *PowerMonitorAndTester* csatlakozni tud. Mind a normál küldés és fogadás és mind a gzip küldés és fogadás szerverét ez a program hozza létre.

4.2.2 DataCreator

A parszoló tesztekhez szükséges fájlok generálásához létrehoztam egy programot, hogy biztosan helyesek legyenek a fájlok és ugyanazokat az adatokat tartalmazzák.

4.2.3 DataProcessor

A *PowerMonitorAndTester* monitorja által készített napló fájlt dolgozza fel, úgy hogy a kapott csv-t egy excel fájllá konvertálja. A konvertálás után a naplóban található futott tesztek külön munkalapon lesznek. A tesztek eredményeit a program automatikusan átlagolja, a tesztek elejéről és végéről pedig levág 1-1 percet.

5 Erőforrások mérése

A mérések több különböző hardveren és körülmények között futtattam, hogy általánosan meghatározhassam az egyes erőforrások és rutinok energiaigényét.
A méréseket úgy végeztem, hogy minden tesztet 12 percig futtattam. Amelyik teszt több lépésből állt (Pl.: Képernyő teszt), azt 24 percig futattam. A tesztek között hagytam 30 másodperc szünetet, hogy a készülék fogyasztása alaphelyzetbe álljon.

Ahhoz, hogy egy készülék pillanatnyi áramfelvételét meghatározhassam, a gyártónak azt támogatnia kell. Amelyik készülék ezt támogatja, azon megtalálható egy fájl, amiből a pillanatnyi áramfelvétel kiolvasható. Ezen fájlok neve és helye készülékről készülékre változik. A fájlok azonosításához a currentwidget[1] projektben lévő listát vettem alapul és egészítettem saját tapasztalataimmal.

5.1 Méréshez felhasznált hardverek

A méréseket a következő készülékeken futtattam:

Típus	Operációs rendszer
Samsung Galaxy Ace 2 I8160	2.3.6
Samsung Galaxy S3 mini I8190	4.1.2
Sony ST25i	4.0.4

3. táblázat felhasznált hardverek

5.2 Szcenáriók

A teszteket különböző körülmények között futattam, hogy általánosan meghatározhassam az egyes erőforrások és rutinok áramfelvételét. Ehhez a következő szcenáriókat hoztam létre:

Szcenárió 1:	
Hálózat	-
GPS műholdak	-
Csatlakoztatott külső eszköz	nincs
Célja	Környezetfüggetlen tesztek futtatására alkalmas, mint például képernyő teszt, hangszóró teszt
Szcenárió 2:	
Hálózat	Jó jelerősség rendelkező hálózat elérhető. Internet nem érhető el a hálózaton, de az adatátvitel tesztekhez használt szerver igen
GPS műholdak	-
Csatlakoztatott külső eszköz	nincs
Célja	Adatátvitel tesztek futtatására alkalmas

Szcenário 3:	
Hálózat	Gyenge jelerősség rendelkező hálózat elérhető. Internet nem érhető el a hálózaton, de az adatátvitel tesztekhez használt szerver igen
GPS műholdak	-
Csatlakoztatott külső eszköz	nincs
Célja	Adatátvitel tesztek futtatására alkalmas
Szcenário 4:	
Hálózat	Átlagos jelerősség rendelkező hálózat elérhető. Internet érhető a hálózaton, de a teszt szerver nem.
GPS műholdak	-
Csatlakoztatott külső eszköz	nincs
Célja	Mindennapi alkalmazások (Facebook, twitter, e-mail) energiaigényének azonosítására alkalmas internettel rendelkező hálózaton.
Szcenário 5:	
Hálózat	-
GPS műholdak	A készülék nem lát műholdat
Csatlakoztatott külső eszköz	nincs
Célja	A GPS modul energiaigényének meghatározása, ha nem lát műholdat és nem tudja meghatározni a pozíciót.
Szcenário 6:	
Hálózat	-
GPS műholdak	A készülék lát műholdat
Csatlakoztatott külső eszköz	nincs
Célja	A GPS modul energiaigényének meghatározása, ha lát műholdat és meg tudja határozni a pozíciót.
Szcenário 7:	

Hálózat	Nincs elérhető hálózat, amire a készülék fel tudna csatlakozni
GPS műholdak	-
Csatlakoztatott külső eszköz	nincs
Célja	Wi-Fi modul energiaigényének meghatározása, ha nem tud hálózathoz csatlakozni.
Szcenário 8:	
Hálózat	-
GPS műholdak	-
Csatlakoztatott külső eszköz	Headset csatlakoztatva
Célja	Headset-en történő hanglejátszás energiaigényének meghatározása

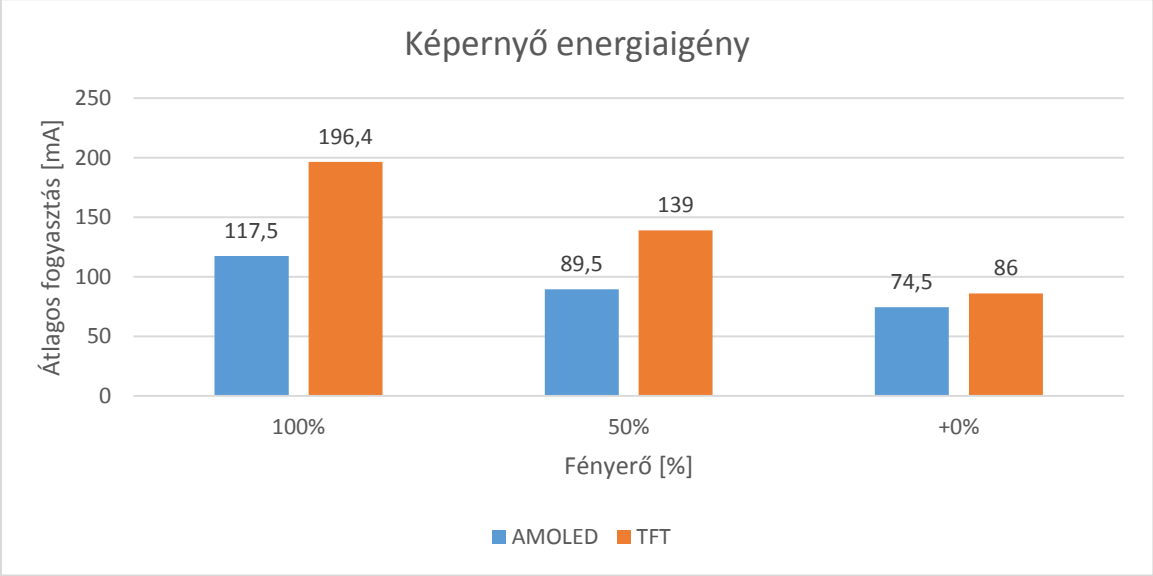
4. táblázat Létrehozott szcenáriók

5.3 Mérési eredmények

A Mérések során keletkezett nyers adatok összegzését a függelék 12.1 fejezete tartalmazza. A következőkben, ebben a fejezetben található adatokat prezentálom és vonok le következtetéseket belőlük.

5.3.1 Kijelző energiafogyasztása

A kijelző a készülékek egyik legnagyobb fogyasztója. Az 5. ábra mutatja be, hogy az energiafogyasztás hogy alakul, a különböző fényerőségeknél. Látható, hogy a kijelző bekapcsolt állapotban fényerő függetlenül is aránylag nagy energiaigénnyel rendelkezik. A fényerő növelésével ez az energiaigény folyamatosan növekszik. Megállapítható, hogy AMOLED kijelzőknél ez a növekedés lényegesen lassabb és minimális fényerejű állapot is valamennyivel kevesebb energiát igényel, mint a TFT kijelzőknél.



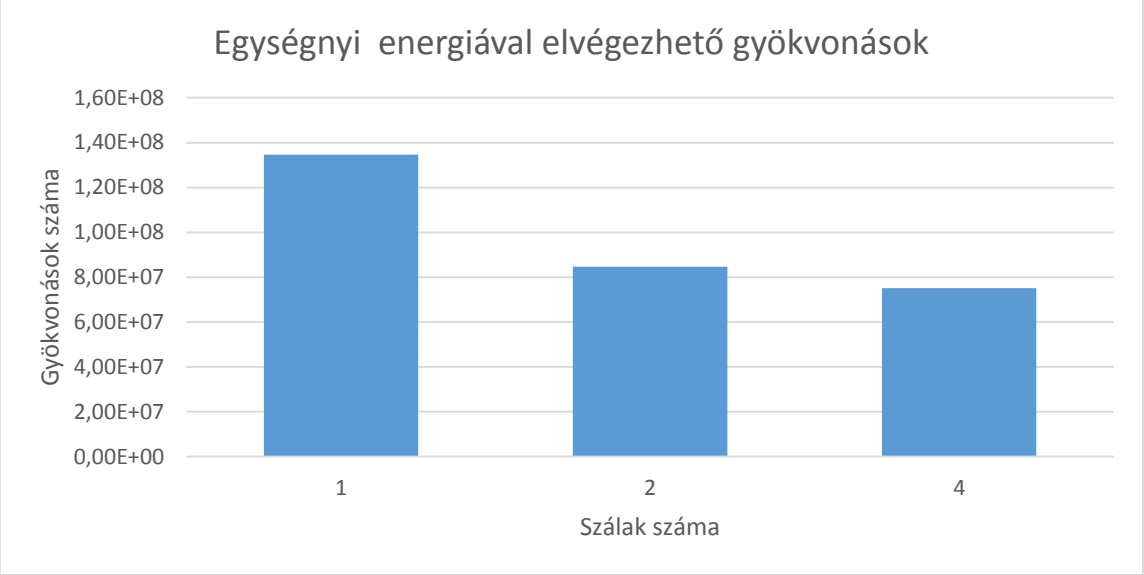
5. ábra Képernyő energiaigénye

5.3.2 CPU energiaigénye

A képernyő mellett a CPU is nagy energiafogyasztó, de az energiafogyasztása szélesebb határok közt mozog. Alacsony (~25%) százalékos, 1 szálas kihasználtság mellett átlagosan 16 mA fogyasztást lehetett mérni. Míg 2 magos processzoron, 4 szálon futtatott folyamat esetén 150 mA-t is elérte a fogyasztás.

A 6. ábra az egységnyi energiával elvégezhető gyökvonásokat mutatja be. Látható, hogy az egy szálon futtatott gyökvonásnak lényegesen kevesebb az energiaigénye, mint a több szálon futtatottnak. Feltehetően az operációs rendszer optimalizációjának köszönhetően ilyenkor csak

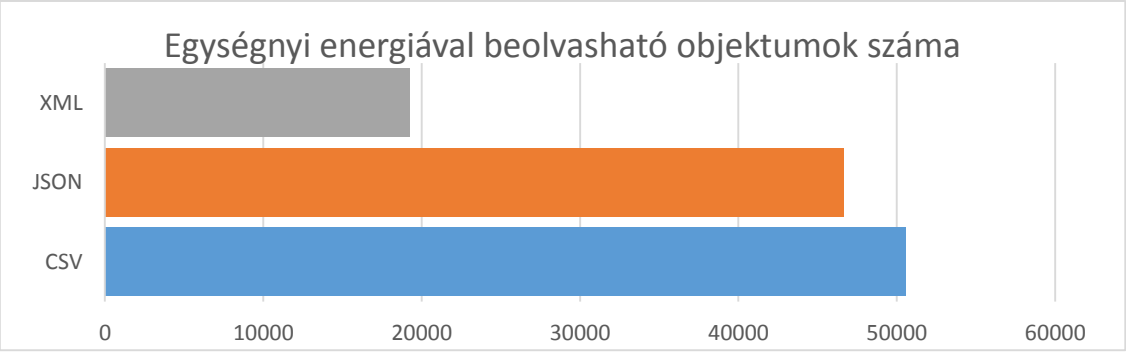
az egyik magot használja, vagy az alacsony kihasználtság miatt csökkenti az alap mag frekvenciát. A két és a négy szálon futtatott gyökvonások energiaigényében is van némi különbség. Ez főként amiatt tudható be, hogy felhasznált hardverekben 2 magos processzorok voltak, így négy szál esetén a szálak ütemezése kontextus váltás nélkül nem oldható meg.



6. ábra Egységnyi energiával elvégezhető gyökvonások száma

5.3.3 Adatfeldolgozás

A fájlból történő adatfeldolgozás energiaigénye egységnyi idő alatt formátumtól függetlenül megegyezik. Ez kb. 120mA, ami lényegében a CPU és némi memóriaolvasás energiaigényének felel meg. Az egységnyi energiával olvasható objektumok számában viszont lényeges eltérések lehetnek. Ezeket az eltéréseket a 7. ábra szemelteti. Látható, hogy lényegesen kevesebb XML objektum olvasható be egységnyi energiával, mint JSON vagy CSV. A legtöbb objektum CSV formátumú fájlból olvasható be egységnyi energiával, de a különbség a JSON-hoz képest nem számottevő. Így ha hierarchikus, könnyebben olvasható formátumot szeretnénk használni, érdemes XML helyett JSON-t választani.

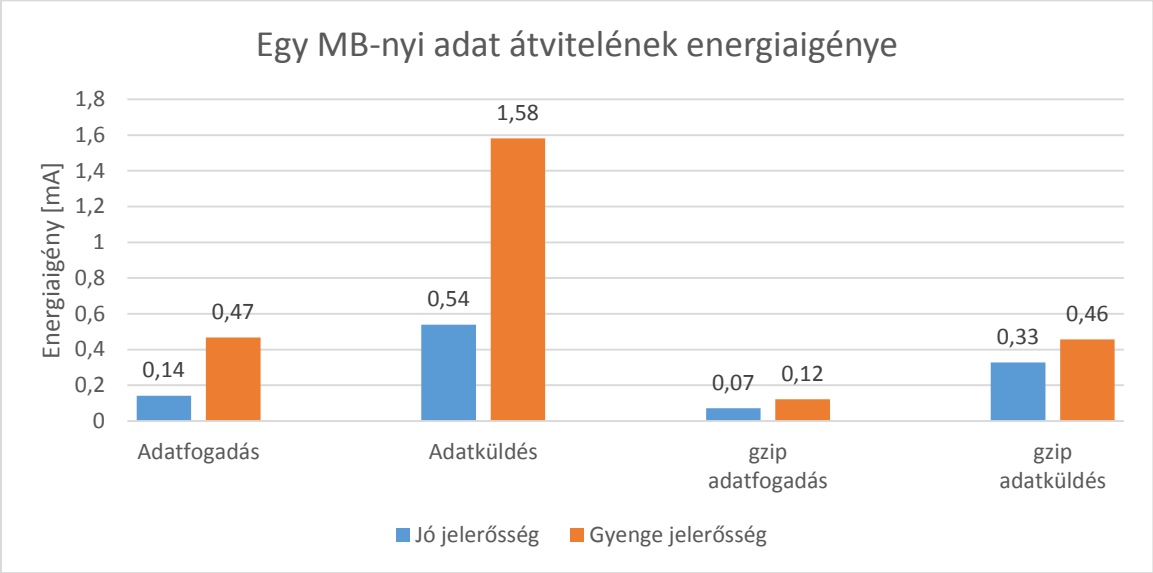


7. ábra Egységnyi energiával beolvasható objektumok száma

5.3.4 Wi-Fi energiaigénye

A Wi-Fi az általam mért erőforrások közül, a készülék legnagyobb energiafogyasztója. Magas terhelés esetén elérheti az átlagos 290mA-s fogyasztást is. A 8. ábra azt mutatja be, hogy egy MB-nyi adat átvitelének mekkora az energiaigénye. Látható, hogy jó jelerősségű hálózaton lényegesen kevesebb energiával lehet ugyan annyi adatot átvinni, mint gyengébb jelerősséggel rendelkező hálózaton. Ez amiatt van, mert adott idő alatt a rádió ugyan annyi jelet képes sugározni, de a rossz csatorna miatt jobb hibajavító kódolást kell alkalmaznia, ami kisebb átviteli sebességgel rendelkezik.

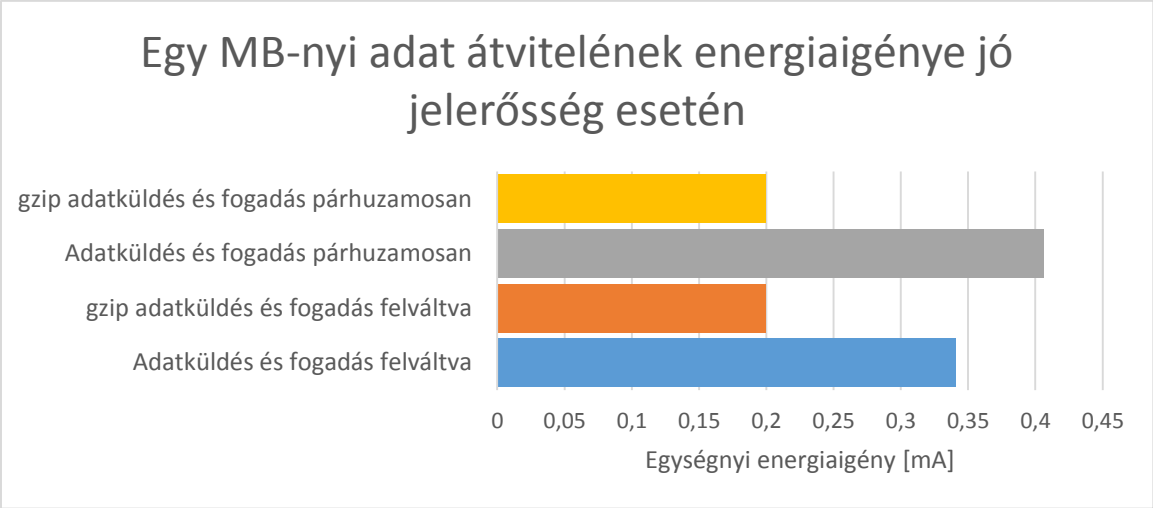
Valamint figyelemre méltó, hogy ha a csatornán átküldött adatot gzip-el tömörítjük nagy fogyasztásbeli csökkenést érhetünk el. Míg sima adatküldés esetén jó jelerősséggel 0,54MB/mA érhető el és rossz jel esetén pedig 1,58 MB/mA, addig gzip-el tömörített adattal rossz jelerősség esetén is csak 0,46 MB/mA, amely jobb eredmény, mint amit sima adatküldéssel, jó jelerősség esetén el lehet érni.



8. ábra Egy MB-nyi adat átvitelének energiaigénye

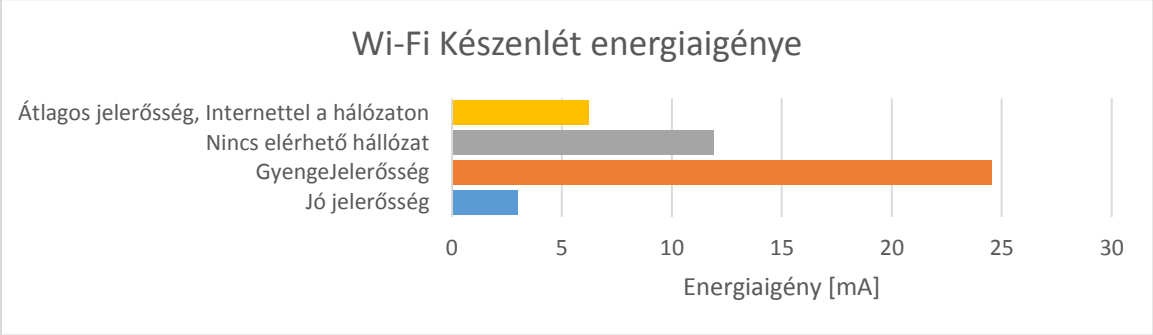
A 8. ábra még szemlélteti, hogy sima adatküldés és adatfogadás estén az erőforrás rosszabbul reagál a jó és gyenge jelerősségbeli különbségekre, mert a fogyasztás átlagosan a háromszorosára nő, míg gzip-el tömörített adatküldés esetén a fogyasztás még a duplájára sem nő.

A 9. ábra azt mutatja be, hogy ha párhuzamosan küldünk és fogadunk adatot vagy felváltva küldünk és fogadunk adatot, akkor az az energiafogyasztásra nincs nagy hatással. Ezt úgy kell érteni, hogy *x* ideig küldünk és fogadunk is párhuzamosan, vagy *y* ideig csak küldünk, majd *x-y* ideig csak fogadunk adatot.



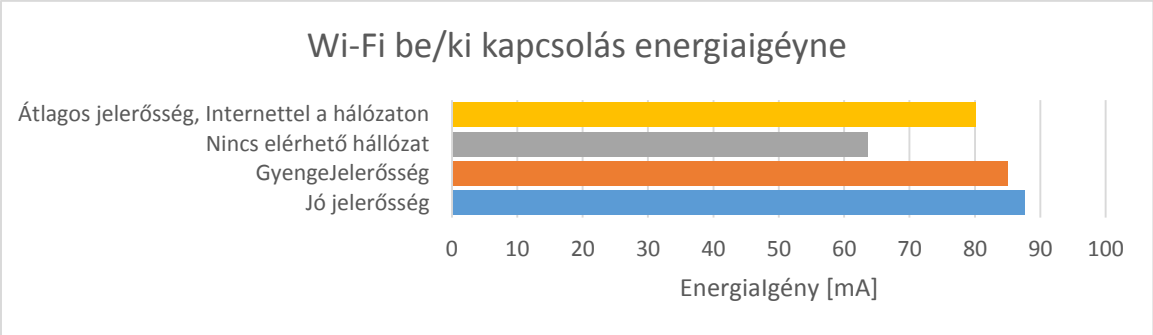
9. ábra Egy MB-nyi adat átvitelének energiaigénye jó jelerősség esetén

Érdemes még megvizsgálni, hogy a Wi-Fi rádió milyen energiaigénnyel rendelkezik, ha nem küldünk hasznos adatot rajta. Ugye hasznos adat küldés nélkül is zajlik némi forgalom a hálózaton, ez az adott protokolloktól függ. Ennek az energiaigényét a 10. ábra szemlélteti. Látható, hogy jó jelerősség esetén nagyon minimális az energia igény, de rossz jelerősség esetén már észrevehető. Érdemes még megjegyezni, hogy ha nincs olyan hálózat a környéken, amire a készülék csatlakozni tud, akkor is nagyobb az energiaigény, mintha egy jó jelerősségű hálózathoz csatlakoztunk volna. Ez főleg a folyamatos hálózat keresések miatt lép fel.



10. ábra Wi-Fi Készenlét energiaigénye

A készenléti energia igény mellett érdemes még a Wi-Fi rádió be/ki kapcsolásának az energiaigényét is megvizsgálni. A 11. ábra bemutatja, hogy az elérhető hálózatok jelerősségétől függetlenül a Wi-Fi rádió bekapcsolásának az energiaigénye átlagosan egyforma. Viszont, ha nincs olyan hálózat a környéken, amire a készülék csatlakozni tudna, akkor a rádió becsapolásának az energiaigénye kevesebb. Ez amiatt van, mert bekapcsolás után, nincs hálózat, amire felcsatlakozna, ezért a kapcsolódással járó energiaigény itt nem lép fel.

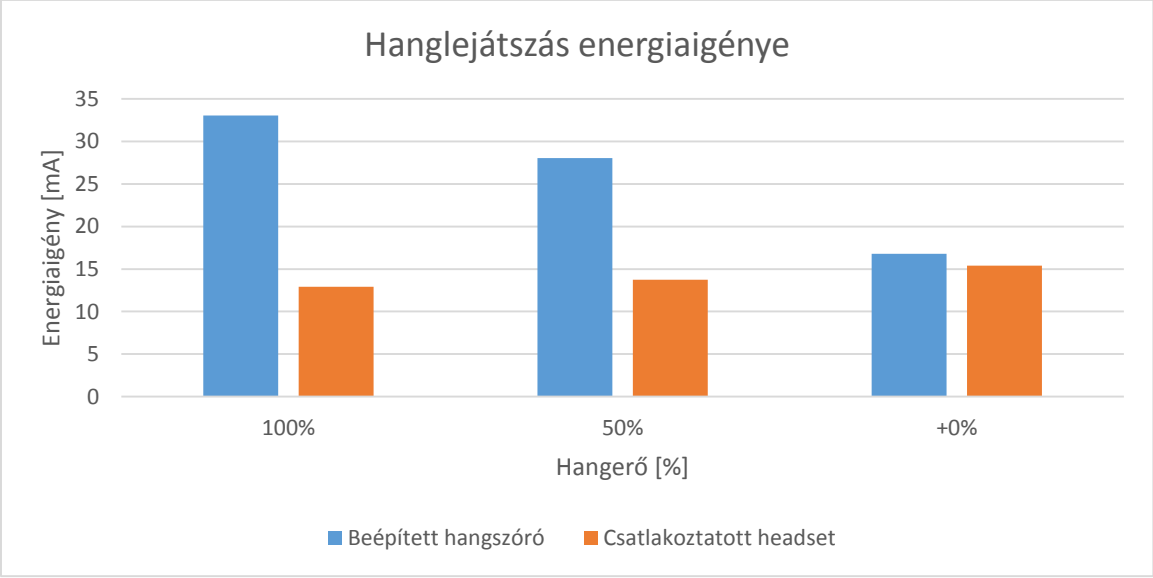


11. ábra Wi-Fi be/ki kapcsolás energiaigénye

5.3.5 Hanglejátszás energiaigénye

A hanglejátszás a többi erőforráshoz képest energiafogyasztás szempontjából nem kiemelkedő. Az 12. ábra szemlélteti, hogy a beépített hangszórón lejátszott hang esetén 100%-os hangerőn átlagosan 33 mA mérhető, míg meglepő módon, ha a hangerő 0%-ra van állítva akkor is 16 mA-s fogyasztás mérhető. A két véglett között a hangerő növelésével az energiafogyasztás is lineárisan nő.

Ha headset van csatlakoztatva a készülékhez, akkor hangerő függetlenül az energiafogyasztás átlagosan konstans 14 mA.

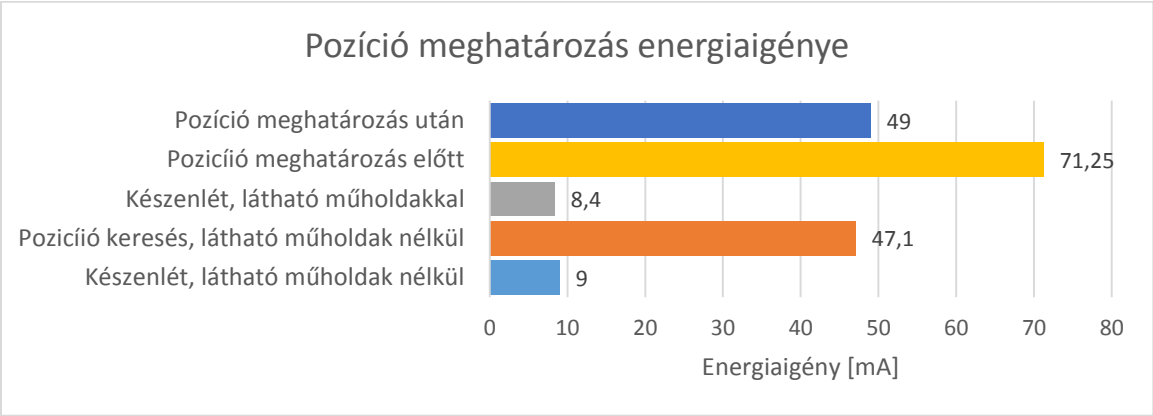


12. ábra Hanglejátszás energiaigénye

5.3.6 GPS energiaigénye

A pozíció meghatározás energiaigénye nagyban változik a körülményektől. A 13. ábra ezt az energiaigényt mutatja be a különböző körülmények közt. Ha a GPS be van kapcsolva, de nem kéri le semmilyen a program a pozíciót, tehát nincs szükség a pozíció meghatározására, akkor is van némi (~8mA) energiaigénye. Ha a készülék látja a műholdakat, az első pozíció

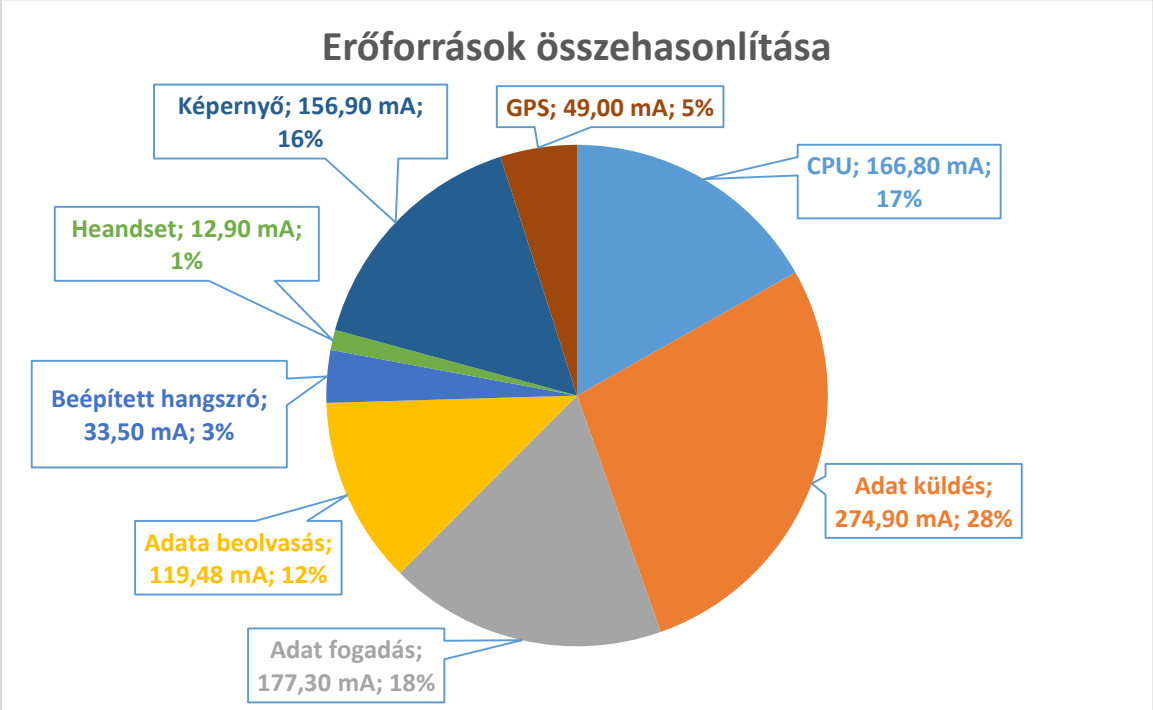
meghatározás előtt van a legnagyobb energiaigénye, átlagosan 71 mA, ha sikerült meghatároznia pozíciót, a pozíció frissítése már csak átlagosan 49 mA-t igényel. Míg, ha a készülék nem látja a műholdakat és úgy próbálja meghatározni az aktuális pozíciót, akkor átlagosan 47 mA-t igényel a GPS működtetése.



13. ábra Pozíció meghatározás energiaigénye

5.3.7 Erőforrások összehasonlítása

A 14. ábra a különböző erőforrások energiaigényét hasonlítja össze. Az egyes erőforrások energiaigényei az adott erőforrás maximális kihasználtságon mért energiaigényét jelneti az ábrán. Látható, hogy a legnagyobb energiaigénnyel az Wi-Fi-n keresztül történő adatküldés rendelkezik. Utána átlagosan egyforma energiaigénnyel az adatfogadás, CPU és a képernyő jön. A legkevesebbet a hanglejátszás és a GPS fogyaszt.

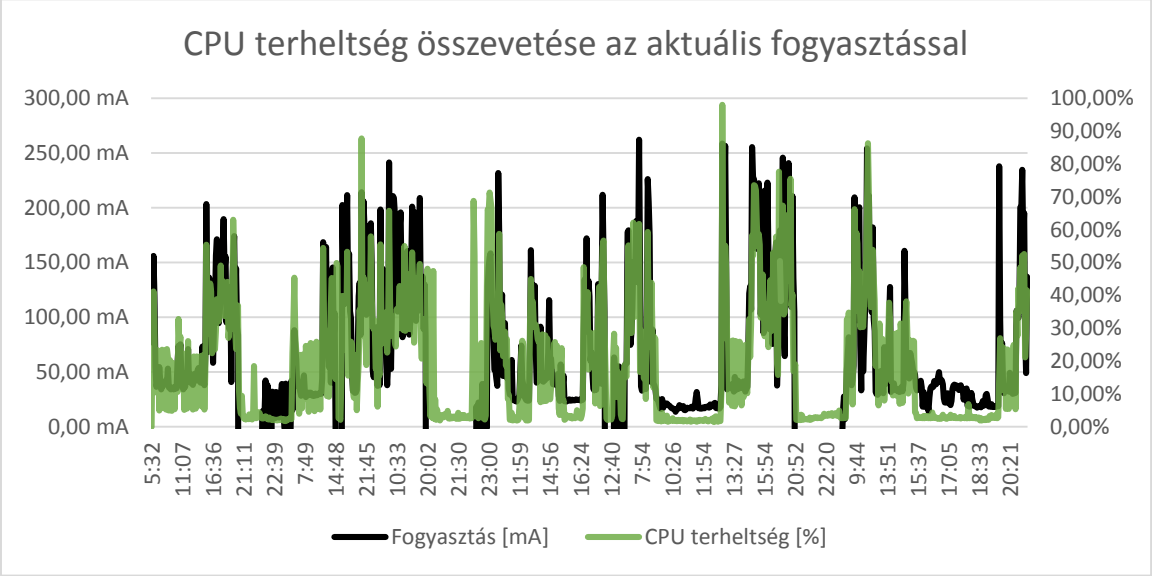


14. ábra Erőforrások összehasonlítása

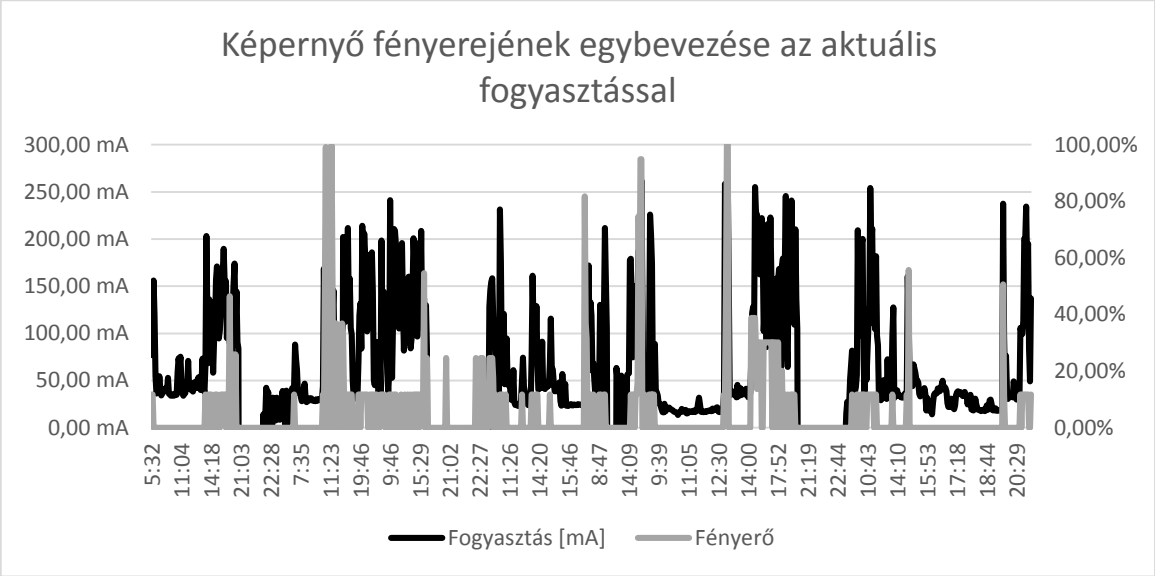
5.3.8 Általános felhasználás monitorozása

Egy I8190 típusú készüléken lehetőségem volt a monitorozó alkalmazás 7 napon keresztül való futtatására. A mért eredményeket a következő 4 ábra mutatják be. Látható, hogy a fő fogyasztók kihasználtsága milyen összefüggésben áll az energiafogyasztással. A CPU terheltsége erősen korrelál a fogyasztással, ezzel szemben a képernyő fényerejének nagysága és a fogyasztás között már nem húzható ilyen egyértelmű párhuzam. Ha az adatküldést és fogadást logaritmikus skálán ábrázoljuk, akkor az így kapott ábrán szintén jól látszik, hogy a megnövekedett Wi-Fi rádió használata megnövekedett energiafogyasztást is von maga után.

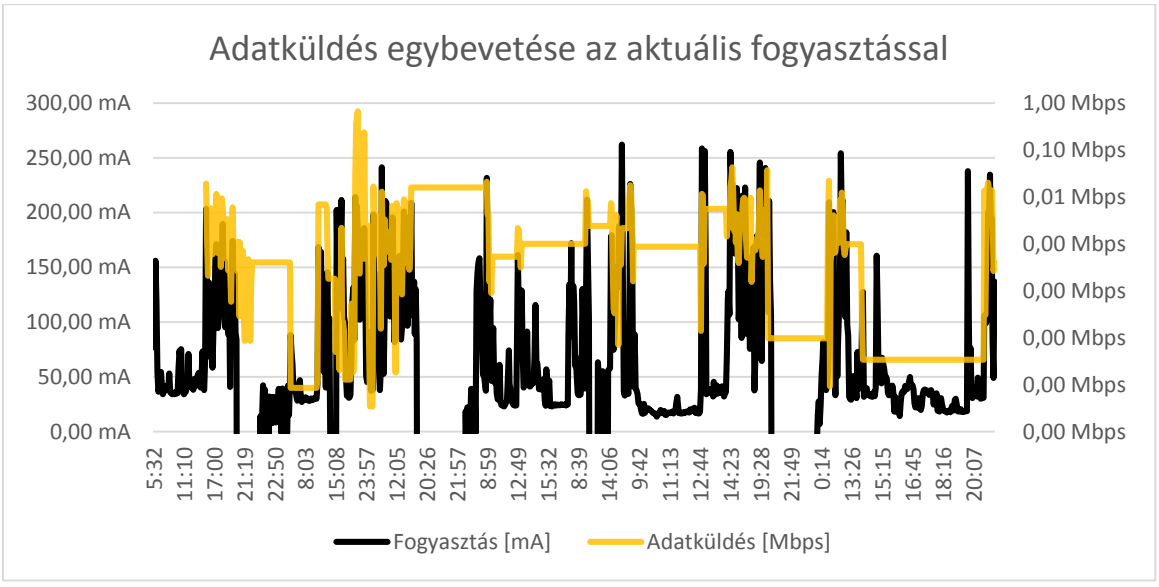
Ezekből megállapítható, hogy a mérések alján a CPU nem a legnagyobb energiafogyasztó, mégis a többi erőforrás működése CPU munkát gerjeszt, ami együtt felelős az aktuális energia felvételért.



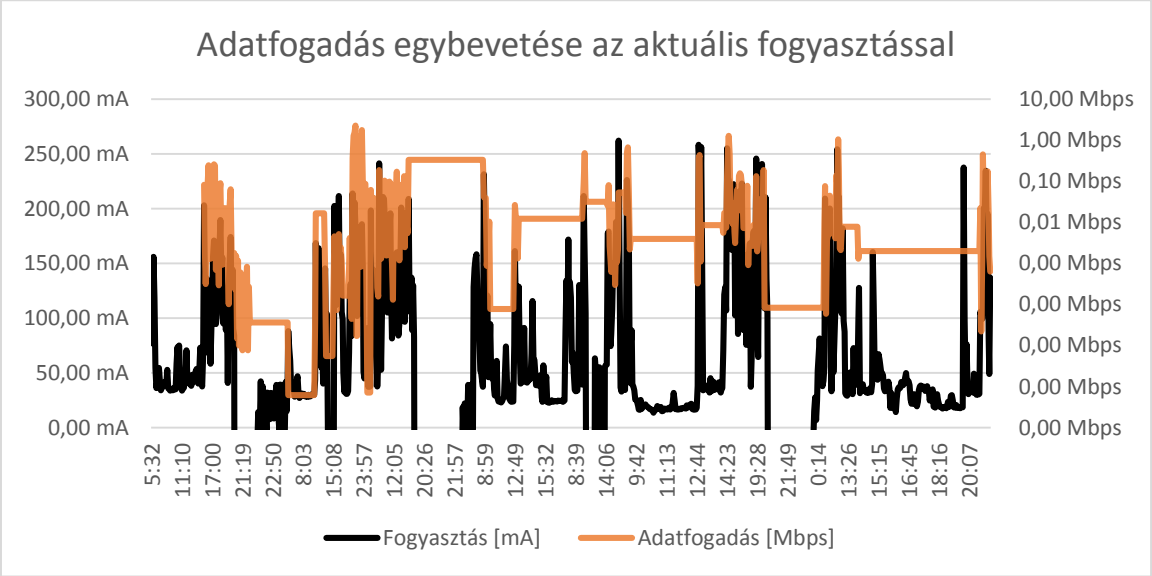
15. ábra CPU terheltség összevetése az aktuális fogyasztással



16. ábra Képernyő fényerejének egybevetése az aktuális fogyasztással



17. ábra Adatküldés egybevetése az aktuális fogyasztással logaritmikus skálán



18. ábra Adatfogadás egybevetése az aktuális fogyasztással logaritmikus skálán

6 Energiahatékony szoftverfejlesztés

Az 5.3.7 fejezetben láthatjuk, hogy a készülékeken a legnagyobb fogyasztók a Wi-Fi, CPU és a képernyő. A GPS és a hangszóró csak minimálisan befolyásolja az energiafogyasztást. Emiatt a fejlesztőknek a Wi-Fi rádióra, CPU és a képernyőre és azok használatára kell a legnagyobb figyelmet fordítaniuk.

6.1 Képernyő optimalizálása

A képernyő használatának csökkentését fejlesztői szemszögből többféleképen tudjuk befolyásolni. Érdemes nagy kontraszttal rendelkező kezelői felületet kialakítani, hogy a felhasználó kisebb fényerővel is kényelmesen tudja használni az alkalmazásunkat. Ez, ahogy az 5.3.1 fejezetben látható, főleg TFT kijelzőknél számít sokat, de AMOLED esetén is befolyásolja az energiafogyasztást. Másik lehetőség, hogy a program állítja be a fényerőt és a képernyő időkorlátját. Ez persze átlagos felhasználói programoknál ellenjavallt. A felhasználót zavarhatja, ha egy program a rendszerbeállításokat megváltoztatja. A fényerő és a képernyő időkorlátjának megváltoztatása a következő kóddal lehetséges:

```
Settings.System.putInt(context.getContentResolver(),
      Settings.System.SCREEN_OFF_TIMEOUT, 0); //időkorlát beállítása 0s-re.
```

```
Settings.System.putInt(context.getContentResolver(),
      Settings.System.SCREEN_BRIGHTNESS, 1); // képernyő fényerejének 1-re
állítása (255 a max)
```

Bizonyos alkalmazások hosszan tartó műveletet végeznek a háttérben, amihez nem mindig fontos a kijelző. Azt, hogy a készülék ne menjen készenléti állapotba és szakítsa meg a folyamatot wake lock-okat használunk. Ezekkel körültekintően kell bánnia a fejlesztőnek, hiszen a készülék, ha nem is végez műveletet, nagyobb energiaigénnyel rendelkezik bekapcsolt állapotban, mint készenlétiben. Minden esetben át kell gondolnia a fejlesztőnek, hogy a művelet elvégzéséhez szükséges-e a képernyő és a billentyűzet, vagy azokat ki lehet kapcsolni. Ha elég a CPU, akkor partial_wake_lock bekapcsolása elegendő, de ha a képernyőt is szükséges a művelethez (például navigálás) akkor már screen_bright_wake_lock szükséges. A következő példa egy wake lock megszerzését és elengedését mutatja be:

```
pm = (PowerManager) context.getSystemService(Context.POWER_SERVICE);
partialWakeLock = pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "Partial
                  Lock"); //partial wake lock objketum létrehozása

partialWakeLock.acquire(); // partial wake lock megszerzése
//...
//műveletek elvégzése
//...
partialWakeLock.release(); // partial wake lock elengedése
```

Fontos, hogy a wake lock-okat, amint nincs rájuk már szükség, azonnal engedjük el.

6.2 CPU optimalizálása

A CPU használat csökkentése is fontos szempont. Az 5.3.2 fejezetben láthatjuk, hogy a CPU energia igénye a terheltséggel növekedésével szintén nő. Ezért próbáljuk a terheltséget minél alacsonyabb szinten tartani hosszú távon. Ezt elérhetjük például azzal, hogy a számításokat felhőben végezzük, ha arra lehetőség van és energiát spórolunk meg. Azzal is csökkentheti a számítási igényét, ha olyan formátumú fájlokat használunk, amit a készülék hardveresen gyorsítani tud, vagy ha a memórián tárolt adatokat nem tömörítve tárolja az alkalmazás. A mai középkategóriás készülékben is általában már több GB tárhely van. Így nem az alkalmazásnak kell a kitömörítéssel foglalkoznia, hanem az adatok már készen állnak. Amennyiben mégis hosszantartó, nagy CPU terheltséggel járó műveletek végzésére van szükségünk, azokat ne futtassuk több szálon, mint amennyi mag fizikailag a rendelkezésre áll, mert a kontextus váltások csökkenthetik a számítás hatékonyságát. És ezeket a számításokat sose a GUI szálban végezzük.

6.3 Adatátvitel optimalizálása

Az 5.3.7 láthatjuk, hogy az adatátvitel igényli a legtöbb energiát. Bár az adatküldés több energia, mint a fogadás, az 5.3.8 fejezetben láthatjuk, hogy az adatküldés átlagban elenyésző a fogadáshoz képest. Ez persze programonként változhat. Wi-Fi-n keresztüli adatot több módon is lehet energiahatékonyan küldeni. Az 5.3.4 fejezetben láthatjuk, hogy a teljes kihasználtságú rádió ugyan annyi energiát igényel jó és rossz hálózati jelerősség esetén is. De jó jelerősség esetén sokkal nagyon throughput érhető el, mint gyenge jelerősségnél. Emiatt érdemes az átviendő adatot összegyűjteni és nagyobb jelerősség esetén átküldeni, ha erre lehetőség van. Természetesen ez egy chat program esetén nem megoldható, de egy zene steamelés esetén letölthetjük a számokat jobb jelerősség esetén is. Az aktuális jelerősség a következő kóddal kérhető le:

```
wifimanager = (WifiManager)context.getSystemService(Context.WIFI_SERVICE);
// WifiManager objektum elkérése
int rssi = 0;
if(wifi.isWifiEnabled()){ // Wi-Fi bekapcsolásának vizsgálata
      rssi = wifi.getConnectionInfo().getRssi();      //aktuális rssi
                                                      //lekérdezése
}
```

Lehetőség van a Wi-Fi rádió szoftveres be/kikapcsolására is. Ez, mint ahogy a fényerő állítása is kerülendő módszer, mert sok esetben zavarhatja a felhasználót. A rádió ki vagy bekapcsolása a következő kóddal tehető meg:

```
wifiManager = (WifiManager) context.getSystemService(Context.WIFI_SERVICE);
// WifiManager objektum elkérése
wifiManager.setWifiEnabled(true); // rádió bekapcsolása
//várakozás, hogy a rádió felkapcsolódjon egy hálózatra
//adatok küldése fogadása
wifiManager.setWifiEnabled(false); // rádió kikapcsolása
```

Adat küldésekor és fogadásakor, ha lehetséges minden esetben tömörített adatot küldjünk és fogadjunk. Gzip stream keresztüli adatküldés minimális extra CPU terheltséggel jár és nagyban csökkentheti a rádión átküldött adatmennyiséget. Az 5.3.4 fejezetben láthatjuk, hogy gzip stream-en keresztül küldött egységnyi adathoz fele, de van, hogy harmada annyi energia szükséges. Gzip-el való adatküldése nem igényel nagy plusz programozói ráfordítást. A következő példakód ezt mutatja be:

```
/*Gzippelt Adatküldés*/
byte[] data; //elküldendő adat
//adat inicializálása
DataOutputStream out = null;
```

```
try {
      out = new DataOutputStream(new
GZIPOutputStream(socket.getOutputStream())); //kimenő stream létrehozása
      out.write(data); // adat elküldése
}catch (IOException ex) { //kivétel kezelése}
finally {
if(out != null){//kapcsolat lezárása
      try {
            out.close();
      } catch (IOException e) {}
   }
}
```

```
/*Gzippelt Adatfogadás*/
byte[] data;
DataInputStreamout in = null;
try {
      in = new DataInputStream (new
GZIPInputStream(socket.getInputStream())); //bemenő stream létrehozása
      in.read(data); // adat fogadása
}catch (IOException ex) { //kivétel kezelése}
finally {
if(in != null){//kapcsolat lezárása
      try {
            in.close();
      } catch (IOException e) {}
   }
}
```

Adatküldéskor érdemes még figyelni arra, hogy ne küldjünk redundáns adatot. Az xml formátum könnyen olvasható, de nagyon sok redundáns adatot tartalmaz. Minden átküldött objektummal átküldjük, hogy az adott objektumnak, hogy hívják az egyes attribútumait, ez sok felesleges adatot jelent. Ezt elég lenne egyszer átküldeni. Ezzel szemben a CSV formátum az adatok neveit csak egyszer küldi el, a fejlécben (vagy ha nincs fejléc, akkor egyszer sem) és utána (az elválasztó karaktert leszámítva) csak tisztán adat van. A 4.1.3.8 fejezetben is látszik, hogy a tesztek során használt objektumból 50000 db CSV formátumban megközelítőleg 4 MB-nyi helyet foglal, Json formátumban 11MB-nyi helyet foglal, míg XML formátumban több mint 15 MB-nyi helyet foglal. Tehát, ha CSV helyett XML formátumban küldjük az adatot, az háromszor akkora forgalmat jelent.

Ha a letöltött adatot lementjük, vagy máshonnan szerezzük be és fájlból olvassuk be, akkor is érdemes figyelni a formátumra. Az 5.3.3 fejezetben láthatjuk, hogy egységnyi energiával körülbelül fele annyi objektumot lehet beolvasni XML formátumú fájlból, mint JSON vagy CSV fájlból.

Továbbá az energiahatékony adattovábbítás eléréshez a Google összeállítatott egy online segédletet „Transferring Data Without Draining the Battery”[10] címen. A segédletben összefoglaltak nagyban tudják csökkenteni a vezeték nélküli adattovábbítás energia költségét.

6.4 GPS optimalizálása

A GPS energiaigényét befolyásolja, hogy mennyi műholdat lát a készülék. Az 5.3.6 fejezetben látható, hogy amíg nem találja meg a készülék az aktuális pozíciót, addig több energiát igényel, majd a pozíció frissítéséhez nem igényel annyi energiát. Ezt szoftverfejlesztő oldalról nem tudjuk befolyásolni. A pozíció változásról kapott frissítéseket viszont befolyásolni tudjuk idő és elmozdulás alapján. Erre a következő kód nyújt egy példát:

```
LocationManager locationManager =
(LocationManager)context.getSystemService( Context.LOCATION_SERVICE );
//feliratokás a pozíció változásra
locationManager.requestLocationUpdates( LocationManager.GPS_PROVIDER, 5,
10, new LocationListener {
// művelet végzése a pozícióval
});
```

A fenti kódban csak akkor kapunk értesítést, ha legalább 5 méterrel megváltozott a pozíciónk és 10 másodperc eltelt az utolsó értesítés óta.

7 Energiatudatos készülék használat

A mérések és a kutatás során több tényezőre is fény derült, melyek alapján megállapítható, hogy tudatos felhasználói viselkedéssel jelentősen csökkenthető a készülék energiafogyasztása.

7.1 Kijelző használata

Az 5.3.7 fejezetben láthatjuk, hogy a Wi-Fi, CPU és a képernyő a három legnagyobb fogyasztó. A felhasználó a kijelző fogyasztását tudj a legjobban szabályozni az 5.3.1 fejezetben látható, hogy a fényerő növelésével a kijelző energiaigénye is növekszik. Sok készülékben található már fényérzékelő, aminek a segítségével a készülék a környezeti fényviszonyok alapján képes változtatni a kijelző fényerejét. Ez a funkció állítható a készülékekben. Érdemes odafigyelni, hogy mindig engedélyezve legyen, mert egyrészről kényelmi funkció, hiszen sötétben automatikusan lekorlátozza a fényerőt, így nem zavaró a szemnek, világosban pedig felemeli, így napközben is jól látható a kijelző. Másrészről pedig a kijelző fényereje sosem lesz magasabb, mint amennyire a felhasználónak a készülék használatához szüksége van. Így készülék sosem fog többet fogyasztania a szükségesnél. Vannak olyan készülékek, amelyek nem rendelkeznek fényérzékelővel. Ebben az esetben a felhasználónak kell kézzel szabályoznia a fényerőt. Az újabb Samsung rom-ok már gyárilag tartalmaznak egy fényerő szabályzó csúszkát az értesítési sávban. Amennyiben nincs ilyen csúszka a készülék értesítése sávjában, érdemes feltelepíteni valamilyen harmadik fél által fejlesztett alkalmazást (Például: Brightness Level), amivel a kezdőképernyőről, kényelmesen és gyorsan lehet állítani a fényerőt. Ez a fényerőszabályzás AMOLED típusú kijelzőknél kevésbé fontos. Az 5.3.1 fejezetben látható, hogy az energiafogyasztás TFT típusú kijelzőknél nagyobb mértékben növekszik nagyobb fényerőnél, mint az AMOLED típusú kijelzőknél.
A fényerősség mellett még a képernyő kikapcsolás időkorlátja is befolyásolja a fogyasztást. Ezért érdemes a képernyő időkorlátját rövidre állítani.

7.2 CPU használata

A CPU használatot nem tudja a felhasználó nagymértékben befolyásolni. Néhány havergyártó (például Samsung) beépíti a romjába egy energiatakarékos módot, amely csökkenti a CPU frekvenciáját. Ezzel lehet korlátozni a CPU használatot, de ez nincs benne minden rom-ban és a harmadik fél által fejlesztett alkalmazások is csak akkor működnek, ha root jogosultságunk van a készüléken.

Az esetek nagy részében a felhasználó két módon tudja csökkenteni a CPU használatból fakadó energia igényt. Az egyik ilyen mód, ha figyeli, hogy melyik alkalmazás használatánál csökken a készülék töltések közti idő és ezeket az alkalmazások használatát elkerüli. Ez sok esetben nem lehetséges, mert a felhasználónak szüksége van az adott alkalmazás által nyújtott szolgáltatásokra. A másik lehetőség, amivel energiát lehet spórolni, hogy a háttérben futó alkalmazásokat (ilyen lehet a zene lejátszó, navigációs szoftver), amiket a felhasználó éppen nem használ, leállítja. Erre léteznek harmadik fél által fejlesztett feladat kezelők, de az alap operációs rendszer is kínál rá lehetőséget. Erre azért van szükség, mert sok esetben, amikor nem látszik egy adott alkalmazás a képernyőn és nem is használjuk a szolgáltatásait (a zene lejátszó nem játszik zenét, a navigáció nem fut), a háttérben még végezhet valamelyik modulja műveletet. Valamint előfordulhat, hogy a wake-lock-ok, amiket az adott alkalmazás lefoglalt nem szabadította fel, így hiába kapcsoljuk ki a készülék képernyőjét, az adott wake-lock megakadályozhatja, hogy a készülék készenléti állapotba menjen át.

7.3 Wi-Fi használata

Az 5.3.7 fejezetben látható, hogy a Wi-Fi keresztül történő adattovábbítás a legenergiaigényesebb. Ezen belül pedig az adatküldés fogyasztja a legtöbbet. Az 5.3.4 fejezetben látható a Wi-Fi használat energiai igénye részletesen. Megfigyelhető, hogy a lényeges különbség van az adattovábbítás energiaigényében, ha jó, vagy ha rossz jelerősségnél továbbítjuk az adatot. Emiatt, ha körülmények adottak érdemes nagyobb adatmennyiségeket (Például: média letöltés) jó jelerősségű hálózaton végezni. A hálózat jelerősségét azért is érdemes figyelni, mert jó jelerősség esetén nagyobb adatátviteli sebesség érhető el, így hamarabb fejeződik be az adott művelet. A fejezetben még látható, hogy a Wi-Fi készenléti állapota is, megnöveli a készülék energiaigényét. Ezért, miután nincs rá szükség érdemes kikapcsolni a Wi-Fi rádiót. Az újabb Android verziók már beépítve támogatják a Wi-Fi automatikus kikapcsolását. A kikapcsolás algoritmusába valamennyi intelligenciát visznek azzal, hogy figyelik a forgalmat, és amíg szükség van a Wi-Fi addig nem kapcsolják ki. Ha az adott operációs rendszer nem támogat ilyen funkciót, érdemes harmadik fél által fejlesztett alkalmazást használni (Pl.: WiFi Auto-Off). Amennyiben az adott szituáció megkívánja az állandó kapcsolatot, készenléti állapotban is érdemes a jelerősségre figyelni, mert gyenge jelerősségű hálózat esetén lényegesen megnövekedhet az energia igény.
A szituáció azt kívánja meg, hogy periodikusan kell küldeni adatot, az egyes küldések közt pedig rövid idő telik el (Például 1 perc), akkor érdemesebb folyamatosan bekapcsolva tartani a

Wi-Fi rádiót, mert a rádió bekapcsolásának lényegesen nagyobb energiaigénye van, mint a Wi-Fi készenléti állapotában fellépő energiaigény.

7.4 GPS használata

Az 5.3.7 fejezetben látható, hogy a GPS modul energia igénye a Wi-Fi-hez vagy a CPU-hoz képest elenyésző, de érdemes figyelmet fordítani rá és ezt az igényt tovább csökkenteni. Az 5.3.6 fejezet bemutatja, hogy a GPS, amikor először határozza meg a készülék pozícióját, addig nagyobb energiaigénnyel rendelkezik. Ezt az igényt tudjuk azzal csökkenteni, hogy ha a készülék támogat A-GPS-t, akkor annak az adatait a GPS használata előtt érdemes frissíteni, mert úgy lényegesen gyorsabban találja meg az aktuális pozíciót.
A GPS modult is érdemes kikapcsolni, ha már nincs rá szükség, több okból is. A háttérben futó szolgáltatások akkor is használhatják a GPS-t, amikor a felhasználó nem is tud róla (Ez sok esetben hibás beállításból ered). Valamint előfordulhat, hogy egy alkalmazás a GPS modul bekapcsolására indul el (akár a háttérben) és fut, amíg használni tudja a GPS-t. Végül biztonsági okok miatt is érdemes kikapcsolni, ha ki van kacsolva a GPS modul, nem tudja egyik alkalmazás se elérni a pontos helyzetünket és így biztos nem tudja megosztani harmadik féllel.

7.5 Hanglejátszás energiatakarékosan

Energiafogyasztás szempontjából a heglejátszás minimális, de bizonyos esetekben mégis érdemes figyelmet fordítani rá. Az 5.3.5 fejezetben láthatjuk, hogy ha a beépített hangszórón játszunk le hangot, annak az energiaigénye a hangerő növelésével nő, míg ha headset-en, vagy más külső eszközön keresztül játszunk le, akkor annak az energiaigénye nem változik jelentősen a hangerő változtatásával. Ezért ha huzamosabb ideig van szükség hanglejátszásra érdemes a készüléket külső hanglejátszó eszközhöz csatlakoztatni (Pl.: Headset, hangfal), hogy csökkentsük a fogyasztását.

7.6 Egyéb energiatakarékossági módszerek

A korábban említett módszerek főleg manuális, a felhasználó által elvégezhető módszerek voltak. A készülék használhatóságának növelése érdekében és az energiahasználat csökkentése érdekében érdemes harmadik fék által készített erőforrás kezelő alkalmazásokat használni.
Ilyen alkalmazás a *Go Launcher EX* által fejlesztett *GO Battery Saver & Power Widget*. Az alkalmazás képes automatikusan kikapcsolni a Wi-Fi rádiót, listát készít a készüléken futó alkalmazásokról és fogyasztás szerint rendezi azokat. Lehet módokat létrehozni, az egyes

módokban külön-külön lehet engedélyezni és szabályozni az egyes erőforrásokat. Idő és akkumulátor állapot triggereket lehet létrehozni a módok közti váltásra. (Például alacsony akkumulátor szint esetén energiatakarékos módra vált). Valamint képes a háttérben futó, nem szükséges alkalmazások automatikus leállítására.

Wi-Fi ki- bekapcsolásra vannak speciális alkalmazások, amelyek pozíció alapján képesek meghatározni, hogy lehet-e elérhető Wi-Fi hálózat a közelben. Ha lehet, akkor bekapcsolják a Wi-Fi rádiót. Ilyen alkalmazás például a *Sebouh Aguehian* által fejlesztett *Smart WiFi Toggler,* amely az aktuális Cella Id-t használja a pozíció meghatározására. Ez lényegesen pontatlanabb, mint a GPS, de nem igényel plusz energiát, mert a GSM modul az esetek nagy részében aktív a készülékeken

8 Összefoglalás, további kutatási irányok

Munkám során a feladatom egy olyan rendszer tervezése és implementálása volt, amely képes a készülék energiafogyasztását szoftveresen mérni és így az egyes erőforrások energia igényét meghatározni. Feladatom volt az így kapott eredmények összegzése és a fejlesztők számára eszközök és módszerek tervezése és készítése, melyek segítik az energiahatékony szoftverfejlesztést, valamint a felhasználók számára egy tudásbázis kialakítása, mely segíti őket a készülék energiatudatos használatában.

A dolgozatomban e feladatoknak eleget tettem. Megalkottam a *PowerMonitorAndTester* Android alkalmazást és hozzá tartozó segédszoftvereket, amelyek képesek a készülék pillanatnyi energiafogyasztását meghatározni szoftveresen. Az alkalmazás képes teszteket futtatni, melyek az erőforrásokat terhelik külön-külön és kombinálva. Így meg tudtam határozni az egyes erőforrások energiaigényét. A dolgozatomban a mért eredményeket összefoglalva, grafikonok formájában mutattam be, amelyekből látható, hogy az egyes erőforrások energiaigénye hogyan változik a különböző körülmények között. Továbbá megmutattam, hogy a három legnagyobb fogyasztó a Wi-Fi, CPU és a kijelző.

Az eredmények alapján a fejlesztők számára energiahatékony szoftverfejlesztést elősegítő eszközöket terveztem és implementáltam, ezeket a dolgozatomban példakódok segítségével prezentáltam. A felhasználóknak pedig egy energiatudatos készülékhasználatot elősegítő tudásbázist hoztam létre, amely felhívja a figyelmet az energiaigényes erőforrásokra és ajánlásokat, ad harmadik fél által fejlesztett programokra, amelyek az energiatudatos készülékhasználatot segítik elő.

A dolgozatom több további kutatási irányt is felvet:

- A munkám során nem sikerült a mobil adatküldés energiaigényét feltérképezni. A későbbiekben lehetne méréseket végezni több mobil technológiával (Edge, 3G, HSDPA stb..) különböző jelerősséggel esetén, cellaváltásokkal. Ez a rész a dolgozatomba a mérések időigényessége és a mérések által megkövetelt pozíció váltások miatt nem került bele.
- A mérési eredményekből kiindulva lehetőség lenne egy olyan alkalmazás készítésére, mely képes intelligens és transzparens módon, akár a felhasználó napi rutinjának feltérképezésével, a készülék erőforrásait úgy ki és bekapcsolni, hogy azok a felhasználónak mindig rendelkezésére álljanak, de a fogyasztást ne növeljék meg szignifikánsan.
  Az alkalmazás biztosíthatna egy olyan api-t, amelyen keresztül a többi alkalmazás energiahatékonyabban lenne képes adatot küldeni. Például az alkalmazás tudná, hogy mikor

várható legközelebb jó jelerősségű Wi-Fi hálózat, a többi alkalmazás, pedig ez alapján el tudná dönteni, hogy megvárja azt az időt, vagy a jelenlegi hálózaton küldi el az adatot.

- A mérések végzése során, néhány hardveren olyan problémába ütköztem, hogy a pillanatnyi energia fogyasztást nem lehetett meghatározni szoftveresen, mert a készülék nem támogatott ilyen funkciót. A továbbiakban a kutatás iránya lehetne egy olyan módszer, amely ezeken a hardvereken is képes lenne pontosan meghatározni az aktuális fogyasztást. Ez a módszer lehetne egy egyszerű alkalmazás vagy akár egy módosított Android rom.

10 Referenciák

[1] Currentwidget, egy energia mérő widget, online: http://code.google.com/p/currentwidget/, 2013.10.21.

[2] Jeff Sharkey. „Coding for Life—Battery Life, That Is", Google IO, 2009

[3] G.P. Perrucci, F.H.P Fitzek, J. Widmer, „Survey on Energy Consumption Entities on the Smartphone Platform" Selected Areas in Communications, IEEE Journal on (Volume:17, Issue: 8), 1999

[4] Niranjan Balasubramanian, Aruna Balasubramanian, Arun Venkataramani „Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications", Internet measurement conference (ICM), 2009 9th ACM SIGCOMM conference, ISBN: 978-1-60558-771-4, New York, USA 2009

[5] I. Kelényi, J. K. Nurminen, Á. Ludányi, T. Ludovszki, „Modeling resource constrained BitTorrent proxies for energy efficient mobile content sharing", Peer-to-Peer Networking and Applications, 2012

[6] I. Kelényi, J. K. Nurminen, Á. Ludányi, „Distributed BitTorrent proxy for energy efficient mobile content sharing", Wireless Personal Multimedia Communications (WPMC), 2011 14th International Symposium, ISBN 978-1-4577-1786-4, Brest, 2011

[7] I. Kelényi, J. K. Nurminen, „CloudTorrent - Energy-Efficient BitTorrent Content Sharing for Mobile Devices via Cloud Services", Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, ISBN 978-1-4244-5175-3, Las Vegas, 2010.

[8] Yuvraj Agarwal Stefan Savage Rajesh Gupta, „SleepServer: A Software-Only Approach for Reducing the Energy Consumption of PCs within Enterprise Environments", USENIX annual technical conference (USENIXATC), 2010, Berkeley, CA, USA 2010

[9] Antti P. Miettinen, Jukka K. Nurminen, „Energy efficiency of mobile clients in cloud computing", 2nd USENIX conference on Hot topics in cloud computing, Berkeley, CA, USA 2010

[10] Optimizing Downloads for Efficient Network Access, Google energiahatékony hálózat elérésről szóló segédlete, http://developer.android.com/training/efficient-downloads/efficient-network-access.html

11 Ábrajegyzék

12 Függelék

12.1 Mérési eredmények

12.1.1 I8190

12.1.1.1 Szcenário 1

Futtatott tesztek		Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]
Rendszer üresjárat teszt		2%	25,5
Képernyő teszt	100% fényerővel		143,9
	50% fényerővel		115,58
	+0% fényerővel		100
Hang teszt	100% hangerő		46,3
	50% hangerő		45,3
	+0% hangerő		39,01
CSV olvasás teszt	575E+04 db beolvasott entitás	52%	131,5
JSON olvasás teszt	562E+04 db beolvasott entitás	54%	135,2
XML olvasás teszt	243E+04 db beolvasott entitás	54%	138,5
1 szálas CPU teszt	016E+08 db elvégzett gyökvonás	27%	44,8
2 szálas CPU teszt	042E+08 db elvégzett gyökvonás	52%	63,7
Összes szálon futó CPU teszt	044E+08 db elvégzett gyökvonás	51%	57,1
2db összes szálon futó CPU teszt	126E+08 db elvégzett gyökvonás	86%	169,9
Összes szálon futó CPU teszt és képernyő teszt	100% fényerő		193,3
	50% fényerő		175,2
	+0% fényerő		175
Összes szálon futó CPU teszt és hang teszt	100% hangerő		140
	50% hangerő		140
	+0% hangerő		134,7
Összes szálon futó CPU teszt, Képernyő teszt és hang teszt	100% hangerő, 100% fényerővel		228,75
	50% hangerő, 50% fényerővel		216
	+0% hangerő, +0% fényerővel		212,5

12.1.1.2 Szcenário 2

Átlagos RSSI a mérés alatt: -21 dB

Futtatott tesztek		**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Készenléti Wi-Fi teszt			24,9
Wi-Fi be/ki kapcsolás teszt			129,6
Adat fogadás teszt	Fogadottadat: 1340 MB Átlagos sebesség: 16,1 MB/s		158,7
Adat küldés teszt	Küldöttadat: 593,4MB Átlagos sebesség: 8,2 MB/s		234,6
Gzip adat fogadás teszt	Fogadottadat: 2,92GB Átlagos sebesség: 10,2 MB/s	34%	167,9
Gzip adat küldés teszt	Küldöttadat: 767,36MB Átlagos sebesség: 3,68 MB/s	47%	177
Adat küldő teszt és adat fogadó teszt	Fogadottadat: 257,52MB Küldöttadat: 37,33 MB Átlagos fogadó sebesség: 2,78 MB/s Átlagos küldő sebesség: 0,6 MB/s	9,6%	179,9
Gzip Adat küldő teszt és gzip adat fogadó teszt	Fogadottadat: 685,5MB Küldöttadat: 121,9MB Átlagos fogadó sebesség: 2,3 MB/s Átlagos küldő sebesség: 0,6 MB/s	20%	216
Adat küldő teszt, adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 896,92MB Küldöttadat: 127,96MB Átlagos fogadó sebesség: 10,5 MB/s Átlagos küldő sebesség: 2,04 MB/s	65,5%	246,7
Gzip adat küldő teszt, gzip adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 1,83GB Küldöttadat: 314,13MB Átlagos fogadó sebesség: 6,44 MB/s Átlagos küldő sebesség: 1,59 MB/s	90%	181,2
Adat küldő teszt, adat fogadó teszt, összes szálon futó CPU teszt, képernyő és hang teszt	100% hangerő, 100% fényerővel		340
	50% hangerő, 50% fényerővel		340
	+0% hangerő, +0% fényerővel		336,2

12.1.1.3 Szcenário 3

Átlagos RSSI a mérés alatt: -88 dB

Futtatott tesztek		Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]
Készenléti Wi-Fi teszt			49,8
Wi-Fi be/ki kapcsolás teszt			126,1
Adat fogadás teszt	Fogadottadat: 424,7 MB Átlagos sebesség: 3,93 MB/s		130,1
Adat küldés teszt	Küldöttadat: 240 MB Átlagos sebesség: 3,3MB/s		247,1
Gzip adat fogadás teszt	Fogadottadat: 2,12GB db Átlagos sebesség: 7,3 MB/s	25%	153
Gzip adat küldés teszt	Küldöttadat: 726,68MB Átlagos sebesség: 3,49 MB/s	48%	205
Adat küldő teszt és adat fogadó teszt	Fogadottadat: 332,9MB Küldöttadat: 53,61MB Átlagos fogadó sebesség: 3,19 MB/s Átlagos küldő sebesség: 0,85 MB/s	10%	169
Gzip Adat küldő teszt és gzip adat fogadó teszt	Fogadottadat: 2,17GB Küldöttadat: 654MB	28%	139
Adat küldő teszt, adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 337,86MB Küldöttadat: 40,4MB Átlagos fogadó sebesség: 3,1 MB/s Átlagos küldő sebesség: 0,64 MB/s	55,3%	216,8
Gzip adat küldő teszt, gzip adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 2,07GB Küldöttadat: 636,4MB	65,6%	188,8
Adat küldő teszt, adat fogadó teszt, összes szálon futó CPU teszt, képernyő és hang teszt	100% hangerő, 100% fényerővel		349,5
	50% hangerő, 50% fényerővel		325,2
	+0% hangerő, +0% fényerővel		319,8

12.1.1.4 Szcenário 4

Átlagos RSSI a mérés alatt: -53 dB

Futtatott tesztek	Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]

Készenléti Wi-Fi teszt	33,4
Wi-Fi be/ki kapcsolás teszt	115,7

12.1.1.5 Szcenário 5

Futtatott tesztek	**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Rendszer üresjárat teszt	2%	31,9
GPS teszt	3%	52,7

12.1.1.6 Szcenário 6

Futtatott tesztek		**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Rendszer üresjárat teszt		2%	41,5
GPS teszt	Pozíció fix előtt	17%	101
	Pozíció fix után	8%	71,2

12.1.1.7 Szcenário 7

Futtatott tesztek	**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Készenléti Wi-Fi teszt		33,32
Wi-Fi be/ki kapcsolás teszt		111,7

12.1.1.8 Szcenário 8

Futtatott tesztek		**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Hang teszt	100% hangerő	18%	36,8
	50% hangerő	17,9%	36,3
	+0% hangerő	18%	35,8

12.1.2 ST25i

12.1.2.1 Szcenário 1

Futtatott tesztek		**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Rendszer üresjárat teszt		1%	11
Képernyő teszt	100% fényerővel	1%	202,6
	50% fényerővel	1%	150,4
	+0% fényerővel	1%	84
Hang teszt	100% hangerő	2%	56,3
	50% hangerő	2%	47,3

	+0% hangerő	2%	31,1
CSV olvasás teszt	611E+04 db beolvasott entitás	49%	139,5
JSON olvasás teszt	560E+04 db beolvasott entitás	50%	141,7
XML olvasás teszt	223E+04 db beolvasott entitás	49%	140
1 szálas CPU teszt	017E+08 db Elvégzett gyökvonás	29%	34,7
2 szálas CPU teszt	046E+08 db Elvégzett gyökvonás	51%	87,5
Összes szálon futó CPU teszt	041E+08 db Elvégzett gyökvonás	54%	68,3
2db összes szálon futó CPU teszt	096E+08 db Elvégzett gyökvonás	86%	163,7
Összes szálon futó CPU teszt és képernyő teszt	100% fényerő	54%	318
	50% fényerő	54%	227,4
	+0% fényerő	54%	123,8
Összes szálon futó CPU teszt és hang teszt	100% hangerő	51%	157,6
	50% hangerő	51%	144,3
	+0% hangerő	51%	136,9
Összes szálon futó CPU teszt, Képernyő teszt és hang teszt	100% hangerő, 100% fényerővel	50%	317
	50% hangerő, 50% fényerővel	50%	258,8
	+0% hangerő, +0% fényerővel	50%	221,1

12.1.2.2 Szcenárió 2

Átlagos RSSI a mérés alatt: -15 dB

Futtatott tesztek		**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Készenléti Wi-Fi teszt		1%	17,4
Wi-Fi be/ki kapcsolás teszt		13%	84,1
Adat fogadás teszt	Fogadottadat: 1,33GB Átlagos sebesség: 13,2 MB/s	30%	223,6
Adat küldés teszt	Küldöttadat: 408,73MB Átlagos sebesség: 5,7 MB/s	17%	306
Gzip adat fogadás teszt	Fogadottadat: 2,63GB Átlagos sebesség: 9,3 MB/s	35%	235
Gzip adat küldés teszt	Küldöttadat: 676,73MB Átlagos sebesség: 3,25 MB/s	51%	296
Adat küldő teszt és adat fogadó teszt	Fogadottadat: 644,37MB Küldöttadat: 50,20MB Átlagos fogadó sebesség: 6,1 MB/s Átlagos küldő sebesség: 0,95 MB/s	16%	221,79

Gzip Adat küldő teszt és gzip adat fogadó teszt	Fogadottadat: 1,37GB Küldöttadat: 198,76MB Átlagos fogadó sebesség: 4,8 MB/s Átlagos küldő sebesség: 1,1 MB/s	34%	265
Adat küldő teszt, adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 674,72MB Küldöttadat: 87,34MB Átlagos fogadó sebesség: 6,2 MB/s Átlagos küldő sebesség: 1,5 MB/s	58%	313
Gzip adat küldő teszt, gzip adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 1,32GB Küldöttadat: 262,85MB Átlagos fogadó sebesség: 4,69 MB/s Átlagos küldő sebesség: 1,3 MB/s	78%	361,3
Adat küldő teszt, adat fogadó teszt, összes szálon futó CPU teszt, képernyő és hang teszt	100% hangerő, 100% fényerővel	60%	522,9
	50% hangerő, 50% fényerővel	64%	536
	+0% hangerő, +0% fényerővel	64%	546,9

12.1.2.3 Szcenárió 3

Átlagos RSSI a mérés alatt: -83 dB

Futtatott tesztek		**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Készenléti Wi-Fi teszt		1%	34,8
Wi-Fi be/ki kapcsolás teszt		13%	80,4
Adat fogadás teszt	Fogadottadat: 220,50MB Átlagos sebesség: 1,8 MB/s	5%	171,6
Adat küldés teszt	Küldöttadat: 132,52MB Átlagos sebesség: 1,8 MB/s	6%	342
Gzip adat fogadás teszt	Fogadottadat: 553,15MB Átlagos sebesség: 1,8 MB/s	8%	179
Gzip adat küldés teszt	Küldöttadat: 581MB Átlagos sebesség: 2,7 MB/s	42%	392
Adat küldő teszt és adat fogadó teszt	Fogadottadat: 179,26MB Küldöttadat: 49,18MB Átlagos fogadó sebesség: 1,4 MB/s Átlagos küldő sebesség: 0,77 MB/s	6,9%	242,5
Gzip Adat küldő teszt és gzip adat fogadó teszt	Fogadottadat: 379,72MB Küldöttadat: 138,36MB Átlagos fogadó sebesség: 1,25 MB/s Átlagos küldő sebesség: 0,72 MB/s	15%	278

Adat küldő teszt, adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 115,50MB Küldöttadat: 58,72MB Átlagos fogadó sebesség: 1,0 MB/s Átlagos küldő sebesség: 0,91 MB/s	50%	352
Gzip adat küldő teszt, gzip adat fogadó teszt és összes szálon futó CPU teszt	Fogadottadat: 334,80MB Küldöttadat: 165,57MB Átlagos fogadó sebesség: 1,1 MB/s Átlagos küldő sebesség: 0,9 MB/s	61%	352,6
Adat küldő teszt, adat fogadó teszt, összes szálon futó CPU teszt, képernyő és hang teszt	100% 100% fényerővel	hangerő, 53,9%	536,79
	50% 50% fényerővel	hangerő, 53,9%	519,9
	+0% +0% fényerővel	hangerő, 54%	505

12.1.2.4 Szcenário 4

Átlagos RSSI a mérés alatt: -50

Futtatott tesztek	**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Készenléti Wi-Fi teszt	1%	15,6
Wi-Fi be/ki kapcsolás teszt	12%	79,15

12.1.2.5 Szcenário 5

Futtatott tesztek	**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Rendszer üresjárat teszt	1%	22,6
GPS teszt	3%	78

12.1.2.6 Szcenário 6

Futtatott tesztek	**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Rendszer üresjárat teszt	1%	11,8
GPS teszt Pozíció fix előtt	4%	78
Pozíció fix után	4%	63.3

12.1.2.7 Szcenário 7

Futtatott tesztek	**Átlagos CPU terhelés [%]**	**Átlagos fogyasztás [mA]**
Készenléti Wi-Fi teszt	1%	26,8
Wi-Fi be/ki kapcsolás teszt	5%	52,8

12.1.2.8 Szcenário 8

Futtatott tesztek		Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]
Hang teszt	100% hangerő	16%	25,5
	50% hangerő	16%	27,7
	+0% hangerő	16%	31,5

12.1.3 I8160

12.1.3.1 Szcenário 1

Futtatott tesztek		Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]
Rendszer üresjárat teszt		2%	11,9
Képernyő teszt	100% fényerő	3%	213,4
	50% fényerő	3%	153
	+0% fényerő	3%	110,6
1 szálas CPU teszt		42%	68,3
2 szálas CPU teszt		93%	132,7
1 szálas CPU teszt és 3 szálas CPU teszt		73%	136,4
Összes szálon futó CPU teszt és képernyő teszt	100% fényerő	85%	328,14
	50% fényerő	85,3%	252,7
	+0% fényerő	85%	230

12.1.3.2 Szcenário 2

Átlagos RSSI a mérés: -36
Az adatküldő és fogadó teszteket 0-val feltöltött tömbbel végeztem

Futtatott tesztek		Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]
Adat fogadás teszt	Átlagos sebesség: 13,42 MB/s	29%	159,3
Adat küldés teszt	Átlagos sebesség: 13,58 MB/s	22%	200
Adat küldő teszt és adat fogadó teszt	Átlagos fogadó sebesség: 13,51 MB/s Átlagos küldő sebesség: 6,0 MB/s	30%	177,4

12.1.3.3 Szcenário 3

Átlagos RSSI a mérés alatt: -88,6 dB

Futtatott tesztek		Átlagos CPU terhelés [%]	Átlagos fogyasztás [mA]
Adat fogadás teszt	Átlagos sebesség: 4,87 MB/s	12,3%	164,8
Adat küldés teszt	Átlagos sebesség: 2,15 MB/s	4%	259,3

Adat küldő teszt és adat fogadó teszt	Átlagos fogadó sebesség: 2,05 MB/s Átlagos küldő sebesség: 1,23 MB/s	7%	232,4