1.0 Bevezetés

1.1 Mobilhálózatok evolúciója

A ma ismert mobilhálózatok, mint minden más technológia világon átment több fejlődési korszakon. Ennek mozgatórugója a technika fejlődése volt, valamint az, hogy az emberiség igényeit technikai elemekkel tudják kielégíteni a Service Providerek. Ahogy nőtt a kényelmi funkció a szolgáltatás mögött, úgy nőtt az érzelmi kötődés is az ügyfelek részéről, ez az a dolog, amit ma egy szolgáltatónak mindenképp bele kell integrálni a szolgáltatásába, hogy az széleskörben eladható legyen.

A ma ismert hálózatok egy újfajta igényt kell, hogy kielégítsenek, amely a negyedik ipari forradalomnak nevezett IoT (Internet of Things) folyamatnak, technológiának köszönhető. Az emberiség legújabb generációjának nagy rész már abba a világba születik bele, amelynek minden egyes szegletében megtalálhatóak az internet álltal nyújtott szolgáltatások. Napi ügyeik intézéséhez minden esetben az internetet használják, ezért nagyon fontos lesz menedzselni azoknak az eszközöknek a hálózati hátterét, amelyek kilátnak rajta keresztül az internetre.

Az Internet of Things egy újfajta szolgáltatásként jelenik meg a Service Providerek életében, amelyet az érzelmi kötődés helyett az olcsóság és energiamegtakarítás oldaláról közelítik meg. Ez az újfajta megközelítés nagyon szükséges és egyben hasznos lépés is volt, amelyet az evolúció generált. Az embereknek nem csak egy eszköze lehet, amely kilát az internetre, egy okos eszközökkel teli világban millió és millió szenzor fog adatot küldeni és kommunikálni egymással. Ezt a fajta leterhelést a mai Internet Service Providerek már nem tudnák kontrollálni a mai mobilhálózatokon, tehát létre kell hozni egy új platformot, hogy teret adjanak a fejlődésnek.

1.2 Mobilálózatok és a WiFi

A lakossági ügyfelek körében is nagy számmal lehet majd értékesíteni ezt a technológiát, de a cégek számára még fontosabb lesz. Kiindulás alap ennél a megállapításnál az, hogy ahol nincsen WiFi ott lehet ugyan úgy internetezni csak mobilhálózaton. Mobilhálózati lefedettség jóval nagyobb, mint a WiFi és megbízhatóbb is annál. Egy olyan országrészen, ahol nincsen elekromos hálózat kiépítve is tudunk adatot küldeni mobilhálózaton.

Míg a WiFi állandó nagy sávszélességet biztosít a végpontoknak, addig az IoT technológián alapuló mobilszolgáltatások kifejezetten alacsony sávszélességet, pont azért, hogy alacsony energiaigénnyel tudjanak pénzt spórolni, az amúgy is kis mennyiségű adat küldésével. Számszerű megközelítéssel élve, ugyan azon a területen nagyságrendekkel több végpontot vagy szenzort tudunk elhelyezni, ha ezeket az új IoT technológiákat használjuk, mintha WiFi rendszert építenénk ki oda. Ráadásul a WiFi rendszer kiépítése még költségeiben is jóval nagyobb lenne.

1.2.a IoT és az ipar

Az IoT technológia nagy változást hoz, hozhat az ipari hatékonyság növelésében. Számszerűen nagyjából 60%-os növekedéssel számolhatunk, amely igencsak pozitív dolog a 21. századi népességnövekedésre tekintettel.

A mezőgazdaságot kiszámíthatóbbá teszi, de az csak egy példa a felhasználhatósága sokszínűségére. Bármilyen ipari vagy közéleti use casebe beépíthető, hiszen fő felhasználási területe a szenzorhálózat kialakítása, amelynek segítségével adatokat gyűjtünk be a világról és ezeknek az adatoknak a tudatában tudjuk optimalizálni az egyes folyamatok végrehajtását. Napra pontosan tudni fogjuk mikor kell betakarítani a gabonát, mikor nem kell meglocsolni a füvet vagy a terményt, mert olyan szenzorhálózat van a segítségünkre, amely képet alkot a természeti jelenségekről. Ez a fajta biztonság es komplexitás fogja előidézni a folyamatok ugrásszerű emelkedését.

Ha tovább megyünk elméleti síkon és elgondolkodunk, hogy mire lehet jó ez a hatalmas mennyiségű adat, amely napi szinten fogja elemezni a világunkat beleütközhetünk egy másik nagyon felfutó pályával bíró ipari forradalmi részvevőbe a mesterséges intelligenciába. A Big Data elemzés felső szintje, ennek eszköze az IoT. Kombinálva a két technológia nyújtotta lehetőségeket nem a szakdolgozatom része, mindenesetre fontosnak tartom megemlíteni ezt is.

1.3 A 2G mobilhálózat

Az 1980-as évek elején bekövetkezett egy ugrás a celluláris telefonhasználatok számában. A nem egységes rendszer következtében viszont nagy eltérések voltak az országokban kiépített rendszerek között. Ennek következtében nem lehetett használni országhatáron kívül a szolgáltatást az analóg rendszerek. E probléma megoldására egy új egységes mobilrendszert alakított ki az Európai Közösség.

Tulajdonságai az új rendszernek:

- Digitális rádiós interface használata
- Jobb minőség, hatékonyság
- Alacsony szolgáltatási költség
- Nemzetközi roaming támogatása
- Új szolgáltatások bevezetésének támogatása

Ennek a rendszernek a segítségével képestek voltunk hangot továbbitani az erre alkalmas eszközeinkön keresztül. A tényleges elterjedése az új rendszernek 1992-ben történt meg Európa szerte, ekkor nagyságrendileg 22 országban volt jelen az ekkor GSM rendszernek nevezett technológia. Ezt követően terjed el a világ minden részére is.

Névleges átviteli sebessége mai szemmel nagyon kicsi volt nagyjából 70 – 230 kbps. De azt a sebességet nem Facebook vagy Instagram használatára tervezték, mint a mai 4G vagy LTE hálózatokat. Annyit viszont fontos megjegyezni, hogy a ma használt IoT technológiák épp ezt az alacsony sebességet használják, tehát olyan mintha egy retró irányt venne a technológia. Persze e mögött jóval komplexebb háttér van, de azért érdemes észre venni a párhuzamot.

Tulajdonsága volt még ennek a hálózatnak, hogy kevés eszközt tudott kiszolgálni azonos területre vetítve, mint a fejlettebbek. Ez a szám nagyjából 100 eszközt jelentett per négyzetkilométer. A késleltetése az adatküldésnek nem volt számottevő 1-5 másodpercet tett ki. Külön sáv volt a feltöltésnek és a letöltésnek, illetve FDD közeghozzáférést alkalmazott.

1.4 A 3G mobilhálózat

A 3G mobilhálózat az evolúciós fejlődés következő lépcsőfoka volt, kibővítve szolgáltatásokkal a már meglévő GSM rendszereket. A 3G képes eszközökkel már volt lehetőség video telefonálni, internetezni és telefonálni is (FDD és TDD is támogatott). Amikor elterjedt ez a fajta technológia sajnos még nem voltak megfelelő eszközök a piacon, hogy az ügyfelek ezt maximálisan kihasználhassák, csak jó pár év múlva jelentek meg az okostelefonok, amik felnőttek erre a feladatra.

A 3G mobilhálózat kiegészíti a már meglévő GSM rendszereket. A mobilhálózatot 3 nagyobb részre osztja fel:

1. Felhasználói készülék (UE)
2. Rádiós hozzáférés (UTRAN)
3. Gyökérhálózat (Core Network)

Az UTRAn feladata, hogy rádiós interface-t biztosítson a core network és a felhasználói készülék között. A részeként funkcionáló Node B-nek hasonló feladata van, mint a GSM rendszerekben használatos BTS-nek, de sok újítással van ellátva ahhoz képest.

- Más modulációt használ, más frekvenciákkal
- Sűrűbben vannak elhelyezve
- FDD és/vagy TDD módon is tud működni
- Teljesítményszabályozása gyorsabb

Névleges átviteli sebesség ezen a hálózaton már nagyságrendekkel volt nagyobb, mint a GSM rendszereknél. Ez a szám 400 kbps és 15 Mbps között tudott mozogni több eszköz is fért el egységnyi területen a GSM-nél kb 100 UE tudott kommunikálni 1 négyzetkilométeren, míg itt 1000 darabra nőtt ez a szám. Az energiafelhasználása az eszközöknek hálózatra csatlakozott állapotban is jóval kisebb lett 0.5 – 1 W körül mozgó értékkel. A késleltetése is lecsökkent az adatküldésnek 100 – 500 ms-ra.

1.5 A 4G mobilhálózat és az LTE

A 3G álltal nyújtott mobil internet nagy sikernek örvendett, ezt felismerve a Service Providerek fejlesztésbe kezdtek, hogy minél több érzelmi kötődést tudjanak nyújtani a felhasználóknak. Ennek a folyamatnak a gyermeke az LTE (Long Term Evolution).

Ennek a hálózatnak a rádiós interface-e felé a legalább 100 Mbps letöltési és 50 Mbps feltöltési sebesség volt. A csomagkésleltetés még kisebb lett (kb 5ms). Dinamikusan tudott sávszélességet választani jellemzően 1,4 MHz, 3 MHz, 5 MHz… 20 MHz értékekkel. Nagyobb mobilitást nyújtott a felhasználók sebességviszonyát is figyelembe véve. Autóban, nagy sebességgel haladva is kapcsolatbontás nélkül tudott kommunikálni a hálózaton keresztül. Nőtt a lefedettsége is a hálózatnak, valamint a cellák szélén is megtalálhatóak voltak azok a tulajdonságai a hálózatnak, mint a centrumában.

A fejlődéseknek hála megnőtt a szolgáltatás minősége a felhasználók felé, több eszközt is ki tudott szolgálni és elvetette az alapjait annak a hálózati forradalomnak, amit az IoT okozott a jövőben.

2. Az Internet of Things (IoT) bemutatása

A 21. század egyik legnagyobb innovációját az infókommunikáció és az erre épülő adatelemzés fogja nyújtani. Ennek a folyamatnak a magját adja az Internet of Things, magyarul a „Dolgok internete”. A jövő városa, okos város, okos autó, okos mobiltelefon. Ezek mind olyan fogalmak, amelyek az IoT keretrendszerekben működnek, működni fognak. A lényege, hogy beágyazott rendszereken alapuló eszközök kommunikációjával modellezhetjük a világot és ebből a hatalmas adatmennyiségből (Big data) van lehetőségünk a mindennapi munkánkat, életünket leegyszerűsíteni, még gördülékenyebbé tenni. Hatalmas mennyiségű eszköz fog kommunikálni egymással a közeljövőben az interneten keresztül, amelynek véges kapacitása miatt felvetődnek alapvető problémák:

- Van ilyen szolgáltatást biztosító hálózat?
- Van olyan eszköz, amely képes használni a hálózatot?
- Biztonságosak lesznek ezek a hálózatok?
- Távolról tudjuk konfigurálni az eszközeinket?

A mobilszolgáltatók evolúciójából kiindulva, ha nincs is de majd lesz a válasz az első kérdésre, a Service Providerek mindig lekövették a világban uralkodó technológiai trendeket, biztosítva platformot a szolgáltatásaik eladására. A szolgáltatás megjelenése után, ha kis késleltetéssel is de mindig megjelentek olyan eszközök, amelyek képesek maximálisan kihasználni annak tulajdonságai.

A szolgáltatás még nem fejlődött ki, de a technológia adott, hogy ilyen IoT mobilhálózatot hozzunk létre. Ilyen a már Magyarországon is bevezetett NB-IoT és a bevezetésre váró Cat-M1 (LTE-M) hálózat is. Mindkettőnek ugyan az a célja a, hogy a jövőbeli kliensek által generált hálózatterhelést redukálják le, de a szolgáltatás minősége maradjon meg, sőt, ha lehet jobb is legyen bizonyos szempontok alapján, mint a mostani mobilhálózatoké. Ez első körben ellentmondásnak hangozhat, de a továbbiakban részletezve lesz a miként je.

2.1 Az IoT szükségessége

Az IoT egy új szintre emeli a mindennapi életet a jövőben, minden területét behálózza majd, és ezáltal segítve annak gördülékenyebb működését. Olyan problémák kezelésére, megoldására fog segítséget nyújtani, amelyre az ember nem képes a túl nagy komplexitás miatt. A világunkat modellezni fogjuk adatok formájában így az sokkal egyszerűbbé válik, lehet rá algoritmusokat írni, kiszámíthatóbb lesz és hatékonyabb.

A jövő IoT technológiái közül néhány példa:

- Okos autó

  Kommunikálnak benne a különböző szenzorok, amelyek modellezik a környezetet, így képes haladni az úton emberi segítség nélkül.

- Okos otthon

  A szenzorok itt is nagyon fontos tényezői a modellezésnek, a mai világban a luxus kategóriának számító igények kielégítésére használják.

- Okos kerékpár

  Nyomon van követve bárhol jár, védve így a lopás ellen, segélykérési opciót lehet beleültetni, sebességet, gyorsulást mérő szenzorral lehet felszerelni.

A fent említett termékek csak töredéke azoknak, amiket be lehet építeni egy IoT keretrendszerbe. A mezőgazdaságon és az okos városokon keresztül az egészségügyben is megtalálhatóak lesznek azok az eszközök, amelyek mind IoT alapon fognak kommunikálni egymással.

2.2 Az IoT és az M2M rendszerek összehasonlítása

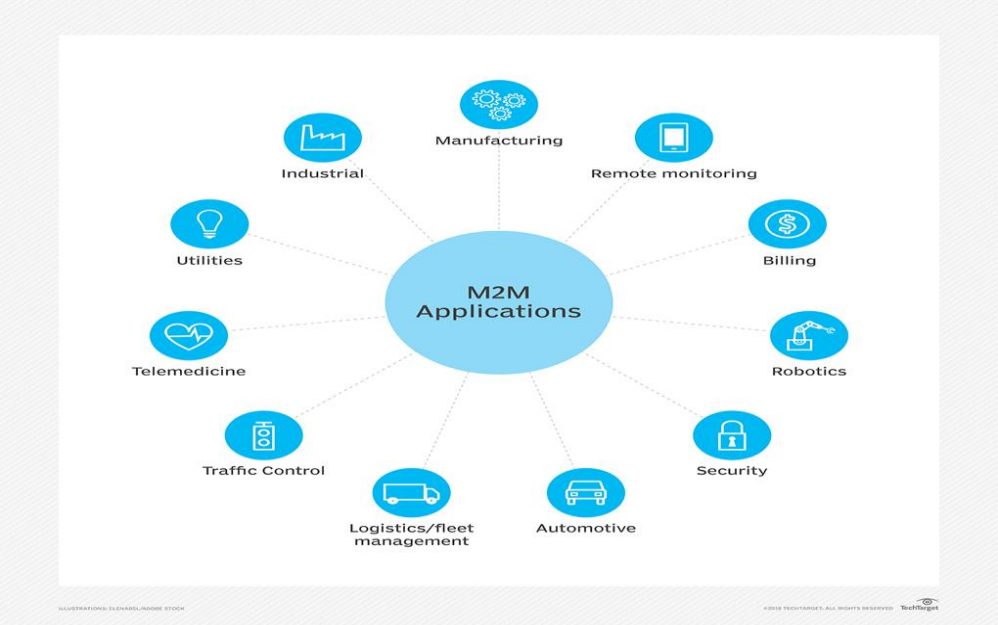
A machine-to-machine technológia nagyon hasonlít az IoT-ra, alapja ennek is, hogy információt cseréljenek különböző eszközök a manuális emberi segítség nélkül. Először a gyártási és ipari környezetben terjed el a használata. Igény merült fel arra, hogy az eszközök működését távolról lehessen megfigyelni és konfigurálni. Ez első M2M technológián alapuló szolgáltatás a mobilhálózaton történő adatküldése volt különböző eszközöknek, ez alapozta meg a mai modern hívóazonosítást is. A Nokia is elsők között ismerte fel az M2M előnyeit az 1990-es években és a 2002-es évekre már szolgáltatást is biztosított az ügyfeleinek a drót nélküli kommunikációra és az okos szolgáltatások igénybevételére.

2003-ban elindítottak egy M2M magazin, amelyben összefoglalták a technológia 6 alapját. Ilyeneket, mint a távoli monitorozás, RFID, szenzoros hálózatok, okos szolgáltatások, telematika és telemetria. Alap célja a szolgáltatásnak, hogy kiszűrje a szenzor adatokat és továbbítsa azokat a hálózaton keresztül. Az M2M rendszereket gyakran használják nyilvános hálózatokon, mint például a mobilhálózaton vagy Ethernet hálózaton, hogy a költséghatékonyságot javítsák.

Az M2M rendszerek fő eszköze a szenzorok, az RFID, a WiFi és a mobil kommunikációs hálózatok, valamint a szenzorokat vezérlő program, amely az eszköz adatainak értelmezésére és az ezeken alapuló döntések meghozatalára szolgál. A rendszerek távoli irányításával az M2M nagyon hatékonnyá teszi folyamatokat:

- Csökkenti a költségeket és minimalizálja a megfigyelt berendezések karbantartását és leállását.
- Növeli a bevételt, feltárva az új üzleti lehetőségeket.
- Növeli a vevő felé a szolgáltatásokat, figyelve az eszközök amortizációját.

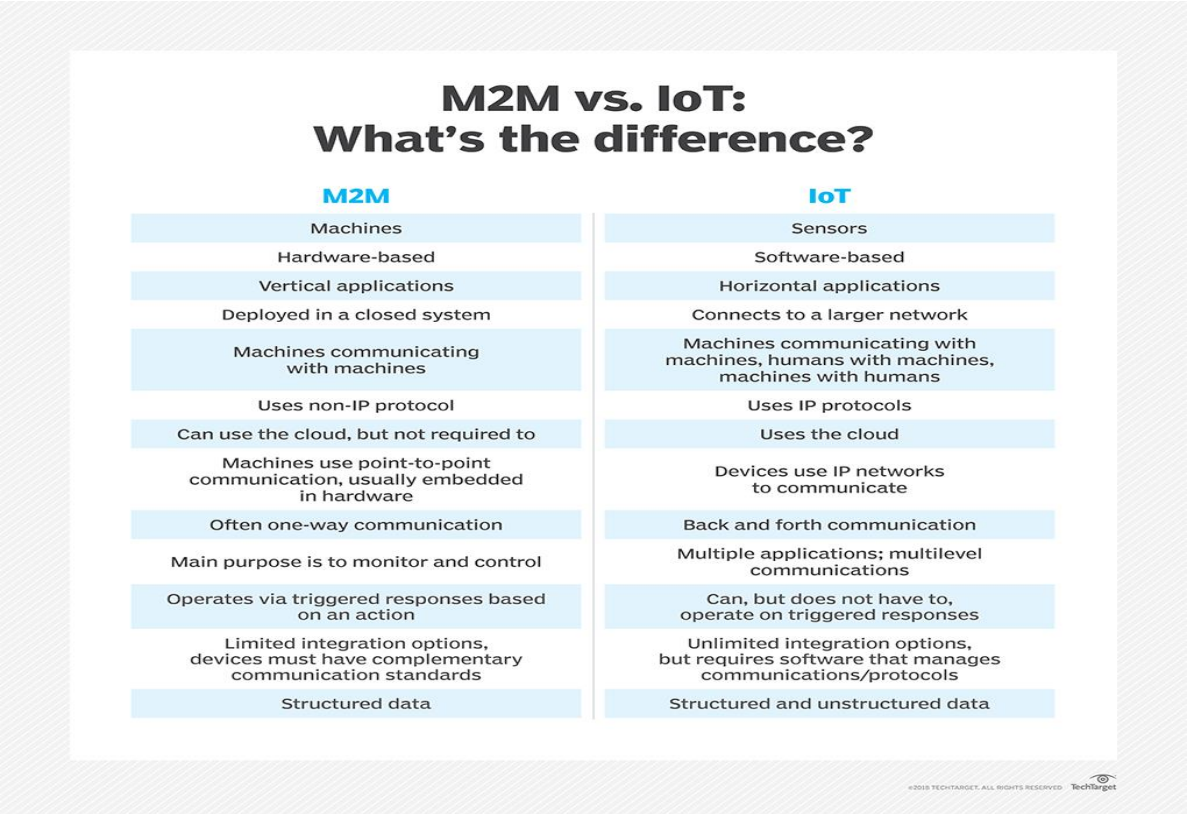
A machine-to-machine sokféle igényt tud kielégíteni, amit az 1. ábra is megfelelően szemléltet



2.2.a IoT és az M2M különbségei

Nagyon sokan használják a két fogalmat egyként, nem különböztetve meg őket. Pedig vannak különbségek a kettő között. Az IoT nem tud M2M nélkül létezni de ez fordítva már elmondható. Mindkét folyamat az összekapcsolt eszközök kommunikációján alapul, de míg az M2M rendszerek gyakran egyedülálló önálló „szigetként" működnek, addig az IoT új szintre emelve ezeket összekapcsolja őket.

Az M2M rendszerek pont a ponthoz kommunikációt folytatnak gépek, szenzorok és hardverek között, mindezt mobilhálózaton vagy kábel alapú hálózaton. Az IoT rendszerek pedig IP alapú kommunikációt folytatnak, összegyűjtve az eszközöket az átjáró funkciót ellátó eszköz számára és feltöltve mindezt egy felhő alapú szolgáltatásba, ahol az adatokat tárolják. Erre az adathalmazra aztán szolgáltatást lehet építeni, és eladni ezt. A fő különbségeket a két technológia között a 2. ábrán szemléltetem.



Az adatok, amelyeket az M2M rendszer összegyűjt, szolgáltatást vezérlő applikációk fogják felhasználni, míg az IoT álltal összegyűjtött adatok egy vállalati rendszert ölelnek fel, hogy javítsák annak a teljesítményét. Míg az M2M a végpontokon van jelen addig az IoT a végpontokat menedzselő keretrendszereket olvaszt egymásba.

2.3 IoT és M2M megoldások a mobil kommunikációs hálózatokban

A mobilhálózati trendek lekövetve a legújabb technológiákat folyamatosan bővítik szolgáltatási termékeiket. A 2020-as évekre a meghatározó vonal a Low Power Wide Area hálózatok lesznek, ami a nevéből adódóan nagy kiterjedésű de alacsony fogyasztású hálózatok megjelenését fogja jelenteni. Az ilyen LPWA megoldásokkal bíró IoT-t támogató mobilhálózatok például a Narrow band Iinternet of Things (NB-IoT) vagy a kicsit fejlettem Cat-M1.

2.3.a LPWA hálózatok a 21. század elején

Az LPWA technológia elterjedését az IoT-nak köszönheti, ugyanis szükség volt egy stabil, nagy területet lefedő és ugyanakkor olcsó, kis teljesítményű hálózat létrehozására, hogy a komplex IoT keretrendszerek működni tudjanak. Ez a vezeték nélküli technológia nyerte el a legnagyobb elismerést az IoT megoldások terén. A hagyományos mobilhálózatok erre nem lettek volna képesek, hiszen ők a már meglévő felhasználókat szolgálják ki, az erre specifikus hálózatukon. Ez a fajta hálózati technológia tehát új fajta megoldásokkal rendelkezik, amelyek az IoT rendszerek igényeit tudják kielégíteni.

A vezeték nélküli kommunikációt két csoportra tudjuk osztani, az egyik, amin hang/adat megy át, a másik pedig amin a gépek kommunikálnak. A hangra és adatra specializálódó hálózatok (2G, 3G, 4G, LTE) nagy áteresztőképességgel rendelkeznek és kis késleltetés

fontos. Azok a felhasználók, akik erre a hálózatra csatlakoznak videókat néznek, zenét hallgatnak, és beszélgetés folytatnak egymással a hálózaton keresztül észrevehető késleltetés nélkül. A gépi kommunikáció alapjaiban különbözik. Az itt átfolyó adatmennyiség nagyon kicsi 1 darab forrásra vetítve, ráadásul az adatküldés gyakorisága sem olyan nagy, mint a hang/adat hálózatoknál. Az adat definíciója nagyon más, annak függvényében, hogy az adatcsere ember-ember vagy gép-gép között történik. Az LPWA technológia a gépi megközelítésre ad kimagasló szolgáltatást. Ezek a tulajdonságok merőben különböznek a hang és adat alapú emberi igényeket is kielégítő hálózatoktól.

Az alacsony energiafogyasztás az első különbség a két hálózat között. A legtöbb ember tudatában van, hogy a mobilkészülékének akkumlátora milyen rendszerességgel merül le és milyen gyakran kell rácsatlakoztatnia a töltőt, hogy a szolgáltatást rajta keresztül élvezni tudja. Az IoT keretrendszerben használatos eszközök szintén mobil kommunikációs hálózatokon keresztül kommunikálnak, szintén szükségük van energiára a működésükhöz. Az IoT eszközök nem olyan mobilis környezetben lesznek felhasználva, mint a mai mobiltelefonok, így az energiapótlás is nehezebben megoldható. Lehetnek olyan felhasználási területtek, ahol a szenzor a föld alatt lesz elhelyezve (parkolószenzor) vagy be lesz építve egy termékbe. A megoldás az energiaproblémára, hogy alacsony fogyasztású szenzorokat használjunk lehetőleg több éves akár 10 éves élettartammal töltés nélkül. Az ilyen szenzorokat el lehet helyezni olyan területekre is, ahol nehezen lenne megoldható az energiapótlás, és mégis képes lesz szolgáltatni a környezetéről adatokat, mindaddig, amíg bírja az akkumlátora. Ez egy erős különbség azokhoz az eszközökhöz képest, amelyek a normál hang/adat hálózatokhoz csatlakoznak.

Azok a technológiák, amelyek nem LPWA alapon működnek limitált lefedettséggel bírnak hozzáférési pontokként. Ezeknél a technológiáknál fontos, hogy közel legyünk a szolgáltatást biztosító egységhez (közel üljünk a routerhez). Az LPWA technológia feloldja azokat a limiteket, amelyeket a WiFi és az eddigi mobilhálózati technológiák vagy akár másik pont a pont közötti kommunikációt biztosító megoldás hordoz magával. Nem minden LPWA hálózat tudja ezeket feloldani, de általánosságban kijelenthető, hogy nagyobb teljesítménnyel bírnak lefedettség terén. A széles lefedettség megkönnyíti azoknak a cégeknek a munkáját, akik különböző és nagy mennyiségű LPWA hálózaton kommunikálni képes eszközt szeretne letelepíteni. Akár több ütemben is, amennyiben a hálózatot bővíteni szeretné, ez nem okoz fennakadást a már üzemben lévő eszközök kommunikációjára nézve.

Az LPWA hálózatok sem tökéletesek, a nagy lefedettségért és a kis energiaigényért az adatküldés késleltetésekor „fizet” a rendszerünk. Ennél az egy esetnél mondható el, hogy a már meglévő mobilhálózatokhoz képest jóval rosszabb mutatókkal bír az LPWA hálózat. Míg egy normál 3G hálózatban ezred másodpercben (ms) mérjük a késleltetést, addig az LPWA hálózatoknál már másodpercekben (sec). Ez nagyságrendekkel nagyobb, tehát a szolgáltatás beteljesülésére többet kell várni. Amennyiben megnézzük a felhasználási területeit a hálózatoknak, akkor ez a hátránya az LPWA hálózatoknak elhanyagolható míg az alap mobilkommunikációs hálózatoknál igen csak fontos. A korábban említett 3G hálózaton elég furcsa lenne olyan beszélgetést folytatni, ahol akár 20 másodpercet is várni kell amíg egyáltalán az üzenetünk eljut a másik félhez, aztán az ő válaszára is 20 másodpercet kell

várni. Márpedig az LPWA ezt tudja csak biztosítani, de ez nála nem hátrány, hiszen például egy mezőgazdasági területen felépített keretrendszerben lévő szenzornál, az, hogy 20 másodperccel később küldi el a talajnedvességre vonatkozó adatot a szervernek az nem okoz problémát, mert ennyi idő alatt hatalmas változás úgy sem következik be. A késleltetés miatt nem lehet mindenhol átállni egy ilyen alapú rendszerre, hiszen van, ahol azonnal szükségünk van a kinyert adatra és másodpercek is fontosak lehetnek a végeredményre nézve.

Az LPWA hálózatok olcsó megvalósítást biztosítanak környezetünk modellezésére, ennek három komponense emelendő ki:

- Infrastruktúra
- Modemek
- Rádió spektrum

Mivel az LPWA hálózatok nagy kiterjedésűek, ezért jóval kevesebb infrastruktúra igényük van, mint a WiFi hálózatoknak. Nem kell sok tízezer routert letelepíteni ugyan azért a lefedettségért és szolgáltatásminőségért. Néhány bázisállomás telepítése elég, amely képes fogadnia rádiós interface-en az adatokat és továbbítani azt a core hálózat felé. Ez a megoldás jóval lecsökkenti a klienseket biztosító cég költségeit is.

A modemek olyan eszközök, amelyek képesek a rádiós interface-en kommunikálni a bázisállomással. Ők képesek továbbítani a szenzor adatokat a mobilhálózat felé. Fontos kritérium, hogy a modem olyan tulajdonságokkal bírjon, ami beleillik az LPWA technológia kereteibe. Túl nagy energiaigénnyel bíró modemet nem érdemes IoT keretrendszerbe beépíteni, mert akkor elveszik az előnye a hálózatunknak. A modem és a szenzor együttese jelenti a kliens oldalt, és képezi az IoT keretrendszer fogalmát. Ezeknek a modemeknek a legyártása, a nagy számú kliens oldali szenzor telepítése alapján olcsó lesz. Jóval olcsóbb, mint egy LTE modem, amelynek az ára nagyjából 40$ körül mozog. Ezeknek a modemeknek az átlagára 3-4$ között fog mozogni. Tehát a cégek ebből a szempontból is képesek lesznek redukálni kiadásaikat.

Az IoT eszközök más fajta adatot szolgáltatnak vagy dolgoznak éppen fel, mint a mindennapokban használt eszközeink. De miben is mások annyira? Az adatforgalom tekintetében biztosan. A most piacon lévő M2M eszközök nagyjából 86%-a kevesebb mint 3MB-ot forgalmaz egy hónapban. Csak belegondolva is, de 3MB-os forgalom egy hónapban egy olyan mobilhálózaton amin telefonálunk elképzelhetetlen. A vállalkozások felismerték az IoT álltal nyújtott hatékonyságukat és ennek a tudatában arra lehet számítani, hogy az LPWA hálózatok fogják uralni az IoT adatforgalom 80%-át a jövőben. Az LPWA hálózatokon, a 3GPP szerint jóval kisebb havi adatforgalom fog mozogni, szerintük a letelepített és már kommunikáló eszközök 70%-a kevesebb mint 33kB adatot fog forgalmazni egy hónapban. Az adatforgalmuk ezeknek az eszközöknek azért ilyen alacsony mert egyszerű szenzorokról beszélünk, amelyek a környezetükről készítenek méréseket és küldik át a hálózaton naponta pár alkalommal. Ők nem küldenek emaileket, töltenek le képeket, hanem néhány byte-nyi adatot küldenek csak át.

Az IoT eszközök lényege, hogy azokat az adatokat megszerezze, amely fontos az vállalkozások számára, mindezt a lehető legolcsóbb módon, a nagy eszközszámra való

tekintettel. Az LPWA technológia már biztosítja az IoT eszközök túlnyomó többségének a megfelelő hálózati kapcsolatot. Amint az IoT még nagyobbra nő annál nagyobb számú kapcsolat fog kiépülni az eszközök között és ezt a leghatékonyabban az LPWA tudja majd lekezelni a mai tudásunk szerint.

2.3.b Narrow Band IoT (NB-IoT) hálózat bemutatása

Szakértők szerint az IoT hatására 2025-re nagyjából 75 milliárd eszköz fog kapcsolódni az internetre. A hálózat növekedésével szükséges a megfelelő támogatás a hálózat részéről minden egyes kliens felé. Ez egyik legjobb LPWA alapú megoldást erre a hatalmas mennyiségű végpontra a Narrow Band IoT hálózata fogja nyújtani, a másik megoldás az LTE-M másnéven Cat-M1 hálózat. A második hálózat fejlettebb, több olyan igényt tud kielégíteni, amit az NB-IoT nem képes, de erről később. Bármelyik hálózatot is fogjuk használni, abban biztosak lehetünk, hogy a kommunikációjuk mobilhálózaton keresztül fog folyni.

A Narrow Band Internet of Things egy költséghatékony, kis energiamennyiséget igénylő (a mostani mobilhálózaton megtalálható eszközökhöz képest), a modulációjából(QPSK) adódóan nagy beltéri lefedettséggel bíró kommunikációt biztosít nagy számú eszköz számára. Maga a hálózat szabványosított keretek között működik, a 3GPP által szabadalmaztatva (Magyarországon). Az LTE sávon belül található In-Band technológiát használ, azaz az NB-IoT vivő egy keskeny spektrumrész, amely az LTE-hez képest 6dB-el nagyobb teljesítménnyel. Az LTE 800 MHz-es sávján belül 180 kHz-nél található az NB-IoT vivő, amennyiben GSM hálózaton keresztül akarjuk használni az NB-IoT-t, akkor a 900 MHz-es GSM sávon belül 200 kHz-nél találhatjuk meg a vivőt. A Narrow Band IoT technológia egyik legnagyobb előnye a nagy beltéri lefedettség. Az alap LTE hálózathoz képest + 15-20 dB-el nagyobb nyereséget tud produkálni. Ennek következtében, bővül a felhasználási területek listája, mint például a betonba épített parkolószenzor, vagy a házfalban lévő hőmérséklet szenzor. Ennek a beltéri lefedettségnek köszönhetően ugyanis képes alacsony jelszintnél is csatlakoznia hálózatra, kommunikálni az eszközökkel, kilátni az internetre. Az NB-IoT hálózat kis sávszélességet biztosít a kommunikáció számára, csupán 200 kbps nagyságrendű elméleti határt éri el. Ez a sebesség a gyakorlatban 20-30 kbps is lehet, tehát nem alkalmas csupán kis mennyiségű szenzoradat küldésére. Elméleti hatótávolsága is nagy, 35 kilométerre becsülik, amekkora távolságból is tud kommunikálni még a rádióállomással.

Az LPWA hálózatok sajátosságai közül a késleltetés, amely merőben több mint a hagyományos mobilhálózatok, de azt a problémát az IoT-nál nem tartjuk számon, hiszen a felhasználási területeknél 20-30es adatkésleltetés még megengedett. Az NB-IoT hálózatnak is sajátossága, hogy magas késleltetéssel bír, ez olykor elérheti a 30 másodpercet is. Tehát a küldő féltől a fogadó csak fél perc elteltével kapja meg az információt. Ez a szenzor adatokat figyelembe véve nem nagy probléma. Az NB-IoT chip, amely a modemen található nagyon kis energiaigényű, eltörpül szinte amellett a vezérlő mellett, amely koordinálja a használatát. A vezérlő az, amely biztosítja a szenzoradatot, hogy azt a modem tovább küldhesse AT parancsok segítségével a hálózaton keresztül a szervernek. A jövőben olyan komplex egységeket kell tervezni, amely képes a vezérlő tulajdonságait és a modemét is egyként kezelni, ez nagyban elősegíti majd az energiaigények csökkentését a keretrendszerre tekintve.

Az NB-IoT rendszerek legfontosabb tulajdonságai tehát, hogy kis adatforgalmat generálnak a hálózaton, ezáltal csökkentve annak a terhelését. Az eszközök, amelyek szükségesek, hogy felépíthessünk egy keretrendszert nagyon olcsók, modem árak már 3-4 euro körül mozognak. A harmadik és egyben legfontosabb tulajdonság, hogy hatalmas mennyiségű kliens tud kapcsolódni egy bázisállomáshoz, ami az IoT szempontjából nagyon fontos jellemző. A ma piacon kapható eszközök elsősorban csak tesztelésre jók, hatékony iparban felhasználható modem sajnos még nem található. A chipsetek is nagyon eltérnek egymástól modemtől függően.

2.3.c A Cat-M1 hálózat alapjai és újdonságai

A Cat-M1 hálózat szintén egy LTE alapú LPWA hálózat, amely megoldásokat nyújt az IoT igényeire. Az előző pontban említett NB-IoT-hoz képest több tulajdonsággal bír, ezek segítségével szélesebb körben lehet majd felhasználni az iparban. Nagyobb sebességet tud biztosítani az eszközök kommunikációjára. Elméleti nagysága ennek 1 Mbps és mindezt 100 kilométeres körzetében a bázisállomásnak tudni fogja biztosítani. A modulációja a QPSK helyett 16QAM lett, ami kicsit több szimbólumot képes továbbítani ugyanazon idő alatt, ebből adódik a sebességkülönbség. A lefedettség szempontjából a tulajdonságokat megtartotta, bel téren kifejezetten jó + 15 dB nyereséget tudunk elérni ilyen hálózatot használva. Míg Európában az NB-IoT terjedt el a közéletben addig Amerikában a Cat-M1 vált dominánssá.

A Cat-M1 hálózathoz való csatlakozáshoz másik fajta modemre is szükség van, ennek a modemnek kicsit nagyobb az energiafelvétele, mint az NB-IoT modemnek, de az energiaért cserébe nagyobb sebességet is kapunk. A nagyobb sebességet fel tudjuk használni olyan területeken, ahol kifejezetten fontos a gyors információcsere. A gyors információcsere az NB-IoT alapú hálózatoknál akadályba ütközött a késleltetés nagysága miatt. Ez a hátrány orvosolva lett a Cat-M1 hálózatnál, LTE hálózathoz szokott késleltetéssel érkeznek meg a csomagok a küldő és a fogadó fél között. A hálózatunk tehát már képes arra, hogy alacsony energiafogyasztás mellett tudjon kommunikálni mobilhálózaton keresztül, egyszerre akár több tízezer eszközt is felölelő területen, nem interferál más hálózatokkal és megbízható end-to-end kapcsolatot biztosít a fogadó és a küldő között. Ezen tulajdonságok lehetőséget adnak egy olyan use case elterjedésének, amely már nem csak az adatot hanem akár hangot is képes torzítatlanl továbbítani az eszközök között. Mivel LTE sávon belül működnek az eszközeink így a VoLTE (Voice over LTE) hívásokat is tudunk rajtuk lebonyolítani, amennyiben a modemünket képesek vagyunk felkonfigurálni erre a célra.

A Cat-M1 hálózat hordozza azokat a tulajdonságokat, mint a Narrow Band IoT kiegészítve a lecsökkentett késleltetéssel, amely egy új dimenziót nyit meg a felhasználási területeinek. Egy autópálya melletti S.O.S telefon-hoz nem kellene kiépíteni állandó áramhálózatot, a kis gyerekeket figyelő mikrofon elemeit nem kellene naponta/hetente cserélgetni, hiszen a fogyasztása ezeknek az eszközöknek drasztikusan lecsökken, amennyiben ezt a LPWA alapú technológiát építik beléjük. Ez csak néhány ötlet, amelyet a világ másik végén Ausztráliában már kereskedelmi forgalomba is hoztak Service Providerek. Idő kérdése tehát, hogy nálunk Magyarországon mikor fog betörni ez a technológia az emberek köztudatába.

3. Az IoT keretrendszer felépítése

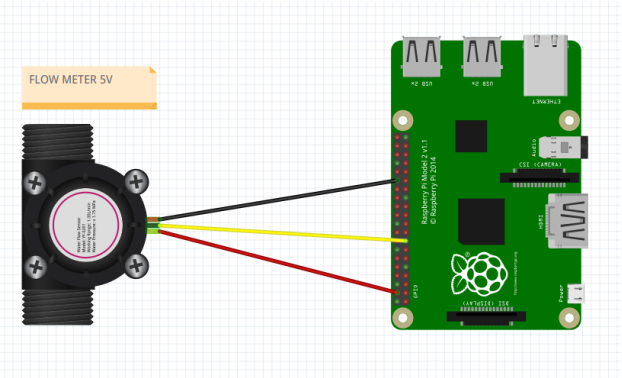
A keretrendszer, amelyet felépítek 4 fő elemből fog összeállni. Fő célja a Cat-M1 hálózaton keresztül történő kommunikáció, adatküldés tesztelése. Tesztelni fogom továbba az ipari felhasználásban fontos szerepet játszó kérdések megoldhatóságát, mindezt egy python szoftver segítségével, amely a vezérlőegységen fog futni. Főbb kérdések, amelyek megválaszolására kitérek majd:

- Távolról elérhető firmware frissítés
- Meglévő konfigurációk elmentése
- Csatlakozási körülmények vizsgálata hiba szempontjából

3.1 A szenzor és az adat

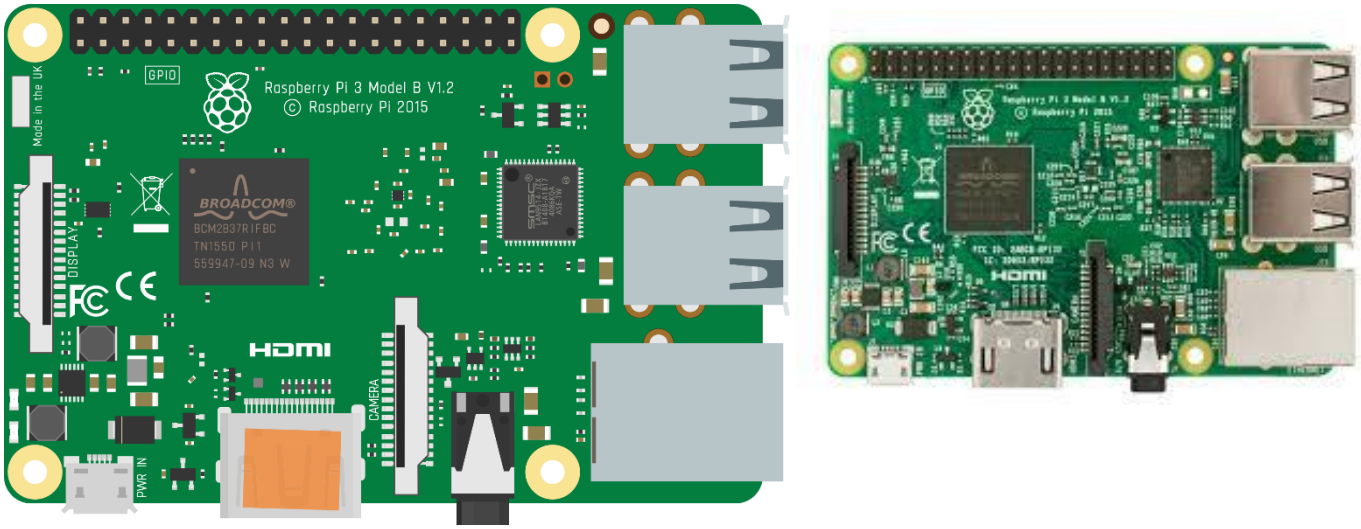
A felépítő elemek közül az első a szenzor maga lesz, amely biztosítani fogja a hálózat felé az értékes adatot. Ezt az adatot fogjuk viszont látni a rendszerünk végpontján egy adatbázisban. A szenzor egy vízátfolyás mérő lesz, amely egy háztartás vízforgalmát fogja modellezni adat formájában. Ennek hatására képesek leszünk vízhasználatunk állapotát figyelemmel kísérni és a fogyasztási szokásainkat megfigyelni, valamint időpontot is tudni fogunk hozzárendelni az adatainkhoz. Ez a felhasználási terület különösen fontos lehet majd a jövőben, de már a mostani helyzetben is, mivel a víz egyre nagyobb érték lesz és az erre irányuló fejlesztésekkel tudjuk hatékonyabbá tenni a víz felhasználásunkat. Ez a módszer az alapja lehet egy fogyasztás szempontjából teljesen automatizált otthon megteremtésére. A kinyert adatok által folyamatos helyzetjelentést kaphatunk rendszereink működéséről, esetleges hibáiról.

A szenzor egy Hall effektuson alapuló YF-S201 víz átfolyásmérő szenzor lesz, amely képes a beépített érzékelőjével mérni a rajt átfolyó vízmennyiséget. A szenzorba épített érzékelő a mágneses tér változása alapján tudja meghatározni, hogy éppen történik e rajta keresztül átfolyás. Az érzékelő a csatornától elzártan található, mivel a víz tönkre tenné. Három vezetéket tartalmaz maga a szenzor, egy pirosat (5V), egy feketét(Föld) és egy sárgát (Adat kimenet). A kiküldött adatok impulzusából kikövetkeztethető az átáramlott folyadék mennyiége. A szenzor specifikációja alapján egy kiadott impulzus 2,25 ml folyadéknak felel meg. Az impulzusok számát megszámolva egyszerű szorzással megkapható az átáramló mennyiség nagysága. A mérési hiba 10% a specifikáció alapján, ez abból adódik, hogy a víz áramlási sebessége eltérő lehet, valamint a kalibrálása is a műszereknek. A kimeneti jel egy négyszögjelként modellezhető, amikor áramlik a folyadék addig a szenzorunk az adat vezetéken keresztül 1-es értéket küld ki, amennyiben nincsen jelen folyadék a rendszerben, úgy 0-ás értéket küld el. A szenzor felépítéséről ábrákat a 3. képen ábrázolom.



3.2 A vezérlőegységek bemutatása

A keretrendszer második eleme egy vezérlő lesz, amelynek feladata, hogy harveresen csatlakozzon hozzá a szenzor és ez alapján szoftveres ki tudja nyerni az adatot belőle, amelyet feldolgozva tovább küld a soros portján a rá csatlakozott modemnek. A vezérlő az én esetemben egy Raspberry Pi 3 Model B lesz. A vezérlőn futó operációs rendszer egy Debian alapú Raspbian. Ez az operációs rendszer az egyik legalkalmasabb környezet, hogy a keretrendszerünket a célnak megfelelően tudjuk menedzselni és konfigurálni. Egyszerűvé teszi a szenzor és a modem hardveres csatlakozását, rendelkezik olyan interfacekkel amelyekre a keretrendszer szempontjából szükség van (GPIO pinek és soros port). A rajta futó operációs rendszer képes futtatni olyan szoftvereket, amelyekkel lehetőség nyílik a hálózatspecifikus problémákat megoldani, illetve kérdésekre válaszokat találni. Támogatja a python fájlok futtatását, és rajta keresztül képesek vagyunk olyan fájlokat tárolni, amelyekkel később elmenthetjük illetve frissíthetjük keretrendszerünk elemeit. A Raspberry tehát keretrendszerünk motorja lesz. Az eszközt bemutató képek a 4. ábrán láthatók.



4. Ábra

A vezérlővel kommunikálva képesek leszünk firmware-t frissíteni a modemen és annak a konfigurációit elmenteni, hogy később felhasználhassuk hibakeresésre illetve konfigurálási alapként is. A vezérlőn keresztül távolról is meg fogjuk tudni szólítani a modemünket, hogy konfigurálni tudjuk.

3.3 A modemek bemutatása

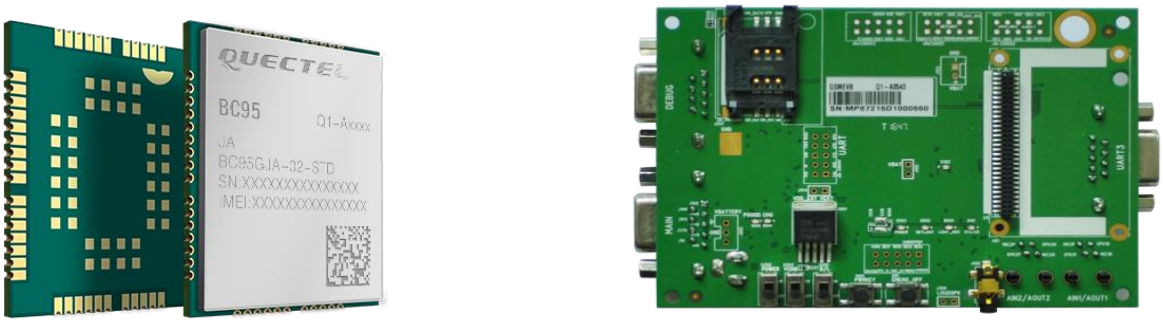
A keretrendszer harmadik eszköze, amelyet bemutatok a modem lesz. Ez az eszköz felelős a rádiós kommunikációért a rendszerben, tartalmazza a SIM kártyát, rendelkezik antennával, amelyen keresztül a mobilhálózat rádiós interface-t el tudja érni, valamint soros porttal, hogy a vezérlővel is tudjon kommunikálni. A soros porton keresztül leszünk képesek konfigurálni is a modemünket, valamint a hálózat aktuális állapotáról is információkat gyűjteni. Többféle modem gyártó cég eszköze különböző módon működik, ez a felhasználási területektől függhet majd, hogy melyiket alkalmazzák. A bemutatandó modemek mind fejlesztői modemek

tekintettel arra, hogy a technológiát még nem követték le a gyártók teljes mértékben. Ezek a modemek arra jók jelen pillanatban, hogy szemléltessük velük az IoT alapú mobilhálózat működését és olyan kérdésekre keressünk velük választ, amelyek az ipari felhasználásban is felhasználhatók lesznek.

3.3.a A Quectel modem

A Quectel gyártó sok egyéb iparban felhasználható modemjei közül a két legfontosabbat fogom kiemelni a BC95-öt és a BG96-ot. Ez a két modem, ami alkalmas arra, hogy a hálózatunkat tesztelhessük velük. A két modem közötti különbség csak annyi, hogy míg a BG96 tud kommunikálni a Cat-M1 hálózaton és az NB-IoT hálózaton is, addig a BC95 csak az NB-IoT hálózatot tudja felismerni. További különbségük az is, hogy a BG96 össze tud kapcsolódni a vezérlővel, olyan formában, hogy átadja neki az IP stackjét. Ebben az esetben nem a modem kap IP címet a hálózattól, hanem a vezérlő. Ezt a megoldást a későbbiekben fejtem majd ki részletesebben.

A BC95 modem tehát az NB-IoT hálózat tesztelésére alkalmas fejlesztői modem. A modem konfigurációja AT parancsok segítségével történik soros porton keresztül. Ezeket a parancsokat automatikusan is kiadhatjuk egy program segítségével, vagy megfelelő kliens oldali szoftvert alkalmazva manuálisan is kiadhatjuk őket. A hálózat tesztelése szempontjából a szoftveres megoldást fogom választani, hiszen ez tud önállóan menedzselni a működést. A modem önmaga egy fejlesztői felületre van ráhelyezve, amely biztosítja a különböző interfaceket a kommunikáció számára. Az 5. ábrán jól látható a modem maga és a fejlesztői felület a különböző interfacekkel. A fejlesztői eszközön kell elhelyezni a SIM kártyát, amely a kliens oldali azonosítás egyik legfontosabb eszköze. Ez tartalmazza azokat az információkat a mobilhálózat számára. A modem nem képes átadni az IP stacket, hogy összeolvadhasson a vezérlővel. Kevés olyan a gyakorlatban használható protokollt támogat, az UDP az, amit magabiztosan lehet vele használni. A socket nyitást és az UDP adat küldését is AT parancsok segítségével kell lebonyolítani, tehát a szoftveres vezérléskor egy scriptben kell lemenedzselni a szenzoradat kinyerését és beágyazását egy UDP csomagba.



5. ábra

A BG96 modem egy modernebb több lehetőséget biztosító eszköz a BC95 modemnél. Abból adódón, hogy képes a Cat-M1 hálózatot is használni, valamint, hogy képes átadni az IP stacket a vezérlőnek. Ezt a működést a legegyszerűbben egy dial-up (betárcsázós) kapcsolatként is megközelíthetjük. A modemet fel tudjuk konfigurálni, hogy amennyiben több hálózatot is érzékel egy adatott helyen (NB-IoT, Cat-M1, GSM), úgy azok között prioritást

tegyen az előre beállított AT parancsoknak köszönhetően. Ha tehát érzékeli az NB-IoT hálózatot és a Cat-M1-et is, be lehet állítani melyikre próbáljon meg először kapcsolódni.

A modem maximálisan 300kbps-os sebességet tud biztosítani hálózattól függetlenül, tehát a Cat-M1 elméleti 1Mbps-os sebességhatárától jóval kisebbet. A széleskörben használható USB, UART, I2C interfaceknek segítségével nagyon egyedire szabható keretrendszereket lehet vele felépíteni. Illesztőprogramjai megtalálhatóak különböző operációs rendszerekre is, mint a Windows, Linux és az Android. Ezekkel a tulajdonsággokkal felvértezve széleskörű M2M szolgáltatásokat lehet biztosítani az ügyfeleknek, akár intelligens mérőműszerek területére is beágyazható. A modemet felkonfigurálhatjuk, hogy ne csak adatot, hanem hangot is képes legyen továbbítani egy másik csatornán a telefonos hálózaton keresztül, ezt a Voice over LTE funkciót, mint hálózat képes ellátni a Cat-M1. Ez a tulajdonsága a modemnek lehetővé teszi, hogy teszteket végezhessünk egy nagyon alacsony költségekkel járó, de mégis jóval stabilabb hálózati tulajdonságokkal bíró hangátviteli megoldás kifejlesztésére, amely az élet nagyon sok területén fontos feladattal bírhat majd.

3.4 A modemek és az IP stack

A modemek, mint a rádiós interface biztosítói rendelkeznek IP stack-el. Ez az előre beprogramozott beállítása a modemeknek hivatott arra, hogy levezényelje a hálózat felé a különböző protokollokon keresztül történő kommunikációt. Ez testesíti meg az OSI modellt a modem szintjén. Ez a modell felelős, hogy különböző eszközök is tudjanak kommunikálni egymással, megszabott szabályok betartásával. Ez az egységes kommunikációs forma alapján tudott elterjedni ilyen széles körben az internet felhasználása. Minden ma ismert internetre csatlakozott eszközbe bele van programozva ez a modell, úgy ahogyan ez a modemekben is benne van.

Két féle módon tudjuk csoportosítani a modemek működését. Az egyik szempont az, hogy a modem képes e átadni az IP stack-et a vezérlőegységének, ekkor a vezérlő és a modem egy nagy egységnek tekinthető. A másik csoport pedig, amikor nem képes átadni és a modemnek saját IP címe lesz. Az első esetben nagyobb lehetőségeink vannak, hiszen a vezérlőn keresztül képesek vagyunk jóval bonyolultabb protokollokon keresztül történő kommunikációra, míg ezeket a funkciókat a mai modemek nem biztos, hogy tudják. Ennek az előnynek hátrányai is van, mert így meg kell fizetnünk a vezérlő és a modem egységes energiafelvételét, ami jóval nagyobb és költségesebb, mintha csak az egyszerű modem kommunikálna állandóan a hálózaton keresztül. Ez a két megoldás két különböző csoportba sorolja a felhasználási területeket, egyik, amikor megengedett az, hogy nagyvonalúak legyünk az energia szempontjából, a másik az, amikor meg spórolni kell vele. Ezt a keretrendszerünk felépítése elején tisztázni kell, hogy a végén ne legyen probléma, a számokat látva és a keretrendszerünk is addig tudja elvégezni a feladatát, ameddig mi szerettük volna.

Az IP stack átadás következménye, hogy tudunk használni olyan protokollokat is, amelyek kifejezetten IoT megoldásoknál hasznosak, mint az például az MQTT.

3.5 A soros port és a 9600 8N1

A modemet és a vezérlőt összekapcsoló interface másnéven soros port, amelyen keresztül tudjuk az utasításainkat átküldeni. Ezen az interfacen megy át minden olyan információ, amelyet kiadunk a vezérlőből és amelyet megkap a modem. Az interfacenek vannak tulajdonságai, amelyeket be kell állítani, hogy a kommunikáció létrejöhessen a két eszköz között. Az IoT keretrendszerben a modem, mint fogadó fél az, aki megszabja például, hogy 1 másodperc alatt mennyi adatot képes feldolgozni hiba nélkül. Ez az információ a fizikai interfacen (soros port) 1-esek és 0-ák sorozatában utazik át, és a fogadó fél ezt konvertálja vissza olvashatóbb formátumra. A 9600 8N1 formátumú adatátvitel egy UART szabvány, amely két eszköz között felépített kapcsolatot reprezentál. A 9600 jelöli a jelzési időt (baudrate), amely az 1 másodperc alatt átvitt adatbitek számát jelöli. A 8N1 formátum jelentése, hogy 8 adatbitet szeretnénk kiküldeni egy keretben (frame) ezt a 8 adatbitet, ahol mindig az LSB (legkisebb helyiértékű bit) kerül először átküldésre az úgynevezett START bit előzi meg. A START bitet követő 8 adatbit után található az opcionális paritás bit is, amely a hiba detektálásához használható. Itt választhatunk páros(E), páratlan(O) és N azaz None paritás módok közül. A paritásbit az előre specifikált formátum szerint állítódik be. Amennyiben páros paritást szeretnénk beállítani úgy az adatbitekben található 1-es bitek számához igazodik, ha ez a szám páros akkor 0-s lesz a paritásbit, amennyiben páratlan, úgy 1-es.

5.4 Az AT parancsok és a modem

A modemekbe épített mikroszámítógépek lehetővé teszik, hogy a modemeket az interface vezetékek helyett szöveges parancsokkal vezéreljük. Ilyen parancskészletet dolgozott ki és szabadalmaztatott az amerikai HAYES cég. A parancskészlet egy alapkészletből és tetszőlegesen bővíthető opcionális készletből áll. Noha a ITU-T ezt az eljárást nem szabványosította, a világon elterjedten alkalmazzák. Minden a terminal által a modemnek küldött parancs (kivéve az A/ és +++ parancsokat) az AT prefixxel kezdődik, melyet a parancs további karakterei követnek. A parancsot a (CR) vezérlő karakter terminálja. Példa erre a szintaxisra az ATDT5044 (CR) parancs, amely parancs utasítja a modemet hívás kezdeményezésre az 5044-es hívószámra. A modem a parancsra számjegyes, vagy szöveges formátumú (OK, RING, ERROR, CONNECT stb.) üzenettel válaszol. A modemek képesek parancs paraméterek hívószámok tárolására is ezeket kikapcsolás után is tárolják nem felejtő memóriában (Non Volatile RAM). A parancsok kiadhatók a terminál emulátor programok monitor módjában, de a legtöbb terminál emulátor program a hívás felépítésével és bontásával kapcsolatos parancsokat menük alá rejti el, így a parancsok kiadása a program menüjéből is kezdeményezhető. A modem különböző válaszokkal reagál a kiadott parancsokra, ezek a következők lehetnek, némelyik specifikus adatot is tartalmaz már.

Szöveg:	Kód	Értelmezés:
OK	0	Sikeres parancs végrehajtás
CONNECT	1	Kapcsolat 300 vagy 1200 bps sebességgel
RING	2	Bejövő csengetés detektálása
NO CARRIER	3	Vivőfrekvencia elvesztése, vagy hiánya
ERROR	4	Parancs error
CONNECT 1200	5	Kapcsolat 1200 bps sebességgel
NO DIAL TONE	6	Nincs tárcsahang
BUSY	7	Foglaltsági hang vétele
NO ANSWER	8	Nincs válasz
CONNECT 2400	10	Kapcsolat 2400 bps sebességgel

Az AT parancsok modemtől függően változhatnak, nem mindegy, hogy milyen chip alapján működik a modem. Az IoT modemekbe kétfajta chipet szoktak beépíteni:

- HiSilicon (Quectel BC95 modem)
- Qualcomm (Quectel BG96 modem)

Minden modemhez tartozik egy részletes specifikáció, hogy melyik chipsettel működik és, hogy milyen parancsokkal képes kommunikálni.

A Quectel által gyártott BG96 eszközének chipsetében egy Qualcomm IoT modem található. Ez a modem felelős azért, hogy a modemet konfigurálni tudjuk, illetve ennek a mikroprocesszornak a feladata, hogy a teljesítményt ezálltal a költségeket is minimalizálja. A modem specifikációja alapján a működésének a célja, hogy LTE modemként tudjon globális kapcsolatokat felépíteni szerte a világban. Alacsony sávszélességet igénylő megoldásokra használható, specifikusan ajánlott a Cat-M1 hálózatokon alapuló IoT megoldások alkalmazására. A késleltetése elhanyagolható, minimális fogyasztással bír, valamint a globális spektrumsávokat is támogatja, ezálltal segítve elő, hogy különböző felhasználási területeken is alkalmazni tudják ezt a terméket. A Qualcomm cég az IoT modemeken kívül más piacokon is nagy innovációt ért el. Chipsetjei megtalálhatóak telefonkészülékekben, táblagépekben és szinte bármilyen „okos” eszközben már.

Az AT parancsokat tehát a modem konfigurációja érdekében adjuk ki. Megmondhatjuk, hogy a rádiós interfacen keresztül milyen hálózatra csatlakozzon, adatokat olvashatunk ki általuk, amelyek a hálózati kapcsolatunk minőségét tudják leírni, illetve a be tudjuk velük azonosítani a végponti eszközünket annak IMSI vagy MSISND száma alapján. Ezeket a parancsokat a modem soros portján tudjuk legegyszerűbben kiadni. Annak módját, hogy a soros portot miként „élesztjük fel” nagyon sok különböző megoldás létezik. Ezek a módok felhasználási szempontból térhetnek el, amennyiben egy keretrendszert építünk és tesztelni szeretnénk a hálózatot úgy használhatjuk a linux rendszereken nagyon elterjed „Minicom” nevű programot, amely egy soros portot nyit a vezérlő és a modem között beállított értékek (baudrate, stop bitek, flow control) alapján. Ez a megoldás arra alkalmas, hogy kis lépéseket, újításokat teszteljünk vele, amelyeket majd beépítünk a szoftveresen futó programunkba. A szoftveres alapon működő soros port konfiguráció lesz az a megoldás, amely már a kész keretrendszerben fog futni, ez a fix programkód fogja menedzselni a modemünk működését, parancsai szolgáltatják a hasznos szenzoradatot és hibakezelése a megfelelő reportok kiadását. Erre a szoftveres feladatra alkalmas programnyelv a python. Megfelelő könyvtárakkal és programokkal rendelkezik, hogy a soros portunkat irányítani tudjuk, amennyiben a minimális működés szintjén maradunk úgy elég egyetlen külső könyvtárát telepíteni az eszközünkre és máris képesek vagyunk virtuális soros portot létrehozni a programkódunkban.

A programunk első feladata tehát, hogy megnyissa a fizikailag már létező soros portot szoftveresen is. A feladat elvégzése után már képesek vagyunk AT parancsokat kiadni a modemünknek, így felkonfigurálni őt a keretrendszer által kirendelt feladatára. Az AT parancsok kiadásának lépései csoportokba osztható:

1. Eszköz adatainak kinyerése
2. Csatlakozás a megadott hálózatra
3. IP cím lekérése

4. Socket nyitása
5. Adatküldés

Az első lépés fontos információkat tartalmaz a végponti eszközünkről. Ezen adatok alapján fogjuk tudni beazonosítani eszközünket a kliens és core oldalról is. Ebben a lépésben használt legfontosabb parancsok a következő pontban lesznek bemutatva.

5.4.a Az eszköz beazonosítása

- **AT + CIMI**
  Ezzel a paranccsal képesek vagyunk lekérdezni a modembe helyezett SIM kártya IMSI (International Mobile Subscriber Identity) számát. Ez a szám egyértelműen tudj azonosítani a SIM kártyánkat a Mobile Core oldalon.
  A parancs kiadásának szintaktikája a 6. ábrán látható módon történik, egyben a lehetséges választ/válaszokat is tartalmazza, valamint a parancs kiadása után megjelenő válasz késleltetését is tartalmazza.

AT+CIMI Request International Mobile Subscriber Identity (IMSI)	
Test Command **AT+CIMI=?**	Response **OK**
Execution Command **AT+CIMI**	Response TA returns **<IMSI>** for identifying the individual (U)SIM which is attached to ME. **<IMSI>** **OK** If there is an error related to ME functionality: **+CME ERROR: <err>**
Maximum Response Time	300ms
Reference 3GPP TS 27.007	

- **AT + CGMR**
  Ennek a parancsnak köszönhetően nagyon fontos információkat gyűjthetünk be a modem szoftveres állapotáról. A modemet az élete során folyamatosan kell fimware updatelni, hiszen a legújabb funkciók csak így érhetők el. Ennek a parancsnak a kiadásával információt kaphatunk arról, hogy éppen melyik firmware van jelen a modemünkön. A parancs szintaktikája a 7. ábrán látható. A parancs válaszában látható lesz, hogy a modem típusa és firmwareszáma mi is pontosan.

AT+GMR Request TA Revision Identification of Software Release	
Test Command **AT+GMR=?**	Response **OK**
Execution Command **AT+GMR**	Response TA reports one or more lines of information text which permits the user to identify the revision of software release. **<revision>** **OK**
Maximum Response Time	300ms
Reference V.25ter	

- **AT + CGMI**
  Ezzel a paranccsal információt kaphatunk arról, hogy melyik gyártó termékét használjuk. Ez különösen fontos lehet akkor, mikor az eszközünk már hosszú ideje működésben van de valami meghibásodott és távolról le kell kérdezni a gyártó nevét, hogy supportot kérhessünk a hiba elhárítására. A válasz a parancsra a gyártó neve lesz. A parancs szintaktikája a 8. ábrán látható.

AT+CGMI Request Manufacturer Identification	
Test Command **AT+CGMI=?**	Response **OK**
Execution Command **AT+CGMI**	Response TA returns manufacturer identification text. **Quectel**

Amennyiben az első lépésben megadott AT parancsok hiba nélkül lefutnak, akkor tovább léphetünk a következő lépésre, ahol már a tényleges hálózati paramétereket adjuk meg a modemünknek, amivel csatlakozni tud a hálózatra. Az első körös parancsokra visszatérve még annyi megjegyzést kell tenni, hogy a hibákat, amelyeket esetlegesen kaphatunk legtöbbször az okozza, hogy nem tettünk be a modemünkbe SIM kártyát, így nem képes kiolvasni az adatokat belőle. Az is probléma szokott még lenni, hogy a SIM kártyát PIN kód védi, ezt amennyiben tudjuk megadhatjuk a megfelelő AT parancs segítségével is, de a tesztelés szempontjából legegyszerűbb, ha beállítjuk a kártyán a modembe helyezés előtt, hogy ne kérjen PIN alapú azonosítást. Fontos megjegyezni itt, hogy a modemre fel kell szerelni már bekapcsolás után a hozzá tartozó antennát, mert ennek hiánya is hibát okozhat.

5.4.b A hálózatra való csatlakozás

A második lépésben tehát hálózati paramétereket fogunk megadni AT parancsok segítségével, ennek köszönhetően a modemünk csatlakozni fog a kívánt hálózatra, lesz IP címe és tudunk rajta keresztül adatot forgalmazni. Ezeket a parancsokat a következő sorrendben kell kiadni a megfelelő csatlakozás érdekében:

1. **AT+QCFG="band",0,80000,80000,1**

   Ezzel a paranccsal megadhatjuk a modemünknek, hogy melyik specifikus frekvenciasávot használja a rádiós interfacen. A Cat-M1-hez használt frekvenciasáv a B20 ezt kikeresve a specifikációból a „80000”-es számot kell megadni a 2. és 3. paraméterében a parancsnak. Az utolsó paraméter annyi jelent, hogy a parancs kiadása után azonnal teljesül a művelet, nem kell külön újraindítani az eszközünket.

AT+QCFG="band" Band Configuration	
Write Command **AT+QCFG="band"[,<gsmbandval>,<catm1bandval>,<catnb1bandval>[,<effect>]]**	Response If configuration parameters and **<effect>** are omitted (that is, only execute **AT+QCFG="band"**), return the current configuration: **+QCFG: "band",<gsmbandval>,<catm1bandval>,<catnb1bandval>** **OK** If configuration parameters are all entered, configure the preferred frequency bands to be searched: **OK** **ERROR**

2. **AT+QCFG="nwscanmode",3,1**

Ezzel a paranccsal meghatjuk, hogy milyen hálózatot keressen (Automatikus, GSM, LTE). Mivel a Cat-M egy LTE alapú mobilhálózat, így a leírás alapján a második paraméternek a 3-as számot kell megadni, hiszen az jelöli az LTE hálózatot. Az utolsó paraméternek ugyan az a funkciója van, mint az előző parancsnál, hogy azonnal beállítsa magának a modem ezt a hálózatot.

Parameter

<scanmode>	Number format. Network mode to be searched. 0 Automatic 1 GSM only 3 LTE only
<effect>	Number format. When to take effect. 0 Take effect after UE reboots 1 Take effect immediately

3. **AT+QCFG="nwscanseq",020301**

Ez a parancs nagyon fontos lesz az ipari felhasználást illetően. Ezzel a funkcióval megadhatjuk, hogy milyen prioritást állítson fel a modemünk azok között a hálózatok között, amelyeket észlel és tudna csatlakozni. Iparban ez fontos nagyon, hiszen egy területen, akár több különböző hálózat is jelen lehet, de ezzel kiszűrjük azt a hibalehetőséget, hogy a rossz hálózatra próbáljon felcsatlakozni. A parancs utolsó paraméterének a feloldása a következő módon tehető meg:

AT+QCFG="nwscanseq" Network Searching Sequence Configuration	
Write Command **AT+QCFG="nwscanseq"[,<scanseq>[,effect]]**	Response If **<scanseq>** and **<effect>** are both omitted, return the current configuration: **+QCFG: "nwscanseq",<scanseq>** **OK**

Parameter

<scanseq>	Number format. Network search sequence. (e.g.: 020301 stands for LTE Cat.M1 → LTE Cat.NB1 → GSM)) 00 Automatic (LTE Cat.M1 → LTE Cat.NB1 → GSM) 01 GSM 02 LTE Cat.M1 03 LTE Cat.NB1

Az itt használt beállítás, tehát először az LTE Cat-M1 hálózatra fog megpróbálni felcsatlakozni, utána az LTE NB-IoT hálózatra és legvégül a GSM hálózatot fogja megpróbálni elérni.

4. **AT+QCFG="iotopmode",0**

Ennek a parancsnak is az a feladata, hogy beállíthassuk rajta keresztül melyik mobilhálózatra csatlakozzon LTE mód alatt. A választható kategóriák a Cat-M1 és az NB-IoT, valamint az is kiválasztható, hogy kifejezetten csak e kettő közül választhasson a modemünk. A parancs utolsó paraméterével beállítottuk, hogy kifejezetten csak a Cat-M1 hálózatra próbáljon meg rácsatlakozni.

AT+QCFG="iotopmode" LTE Network Search Mode Configuration	
Write Command **AT+QCFG="iotopmode"[,<mode>[,<effect>]]**	Response If **<mode>** and **<effect>** are both omitted, return the current configuration: **+QCFG: "iotopmode",<mode>** **OK** If **<mode>** and **<effect>** are not omitted, configure the network category to be searched under LTE network mode: **OK** **ERROR**

Parameter

<mode>	Number format. Network category to be searched under LTE network mode. 0 LTE Cat.M1 1 LTE Cat.NB1 2 LTE Cat.M1 and Cat.NB1
<effect>	Number format. When to take effect. 0 Take effect after UE reboots 1 Take effect immediately

5. **AT+QCFG="servicedomain",1,0**

A parancs kiadását követően beállítható a modem azon funkciója, hogy milyen szolgáltatás tartalmakat lehessen rajta keresztül elérni. Mivel a modemünket Cat-M1 hálózaton szeretnénk működteti, ami pedig LTE alapú így a választható PS, illetve CS csoportok közül nekünk a PS funkciót kel választanunk. A parancs kiadását követően a modem újraindítása szükséges, hogy a beállított funkciók végrehjtódjanak.

Parameter

<service>	Service domain of UE 0 CS only 1 PS only 2 CS & PS
<effect>	Number format. When to take effect. 0 Take effect after UE reboots 1 Take effect immediately

6. **AT+CGDCONT=1,"IP","APN"**

Az APN egy nagyon fontos része a mobilhálózatnak, ezen keresztül képesek a végpontok kommunikálni az internettel, rajtuk keresztül folyik át az adatmennyiség az eszközünk felé. Ezért fontos beállítani, hogy melyik APN-t is használjuk a faladatunk elvégzésére. A Cat-M1 egy LTE alapú mobilhálózat, tehát nem lesz szükség külön APN bevezetésére, használhatjuk az eredetit, amely minden telefonban megtalálható. Az így létrehozott kapcsolat az APN és a végpont között már lehetőséget nyújt arra, hogy kapcsolatot teremthessenek az IoT eszközeink egymással, de fontos megjegyezni, hogy minden egyes eszközön meg kell tenni ezt a beállítást a megfelelő működés érdekében. A parancs kiadásával aktiváljuk a modemnél a Packet Data Procotol-t, amely felelős azért, hogy a modem felől kiküldött adatokat eljuttassa a hálózat felé. Felosztja szegmensekre a kiküldendő adatot, majd a végponton újra egyesíti azt, így biztosítva a megbízható adatkapcsolatot. Ez a protokoll az alapja bármilyen kommunikációnak a mobil core felé, így nagyon fontos jelentőséggel bír a mi keretrendszerünk esetében is.

AT+CGDCONT Define PDP Context	
Test Command **AT+CGDCONT=?**	Response **+CGDCONT:** (range of supported **<cid>**s), **<PDP_type>**, **<APN>**, **<PDP_addr>**, (list of supported **<data_comp>**s), (list of supported **<head_comp>**s) **OK**
Read Command **AT+CGDCONT?**	Response **+CGDCONT:** **<cid>,<PDP_type>,<APN>,<PDP_addr>,<data_comp>,<head_comp>[...]** **OK**

A típusa a kapcsolatnak többféle lehet, nekünk az IP alapú a megfelelő ezért ezt kell megadnunk a második paraméterében a parancsnak. A többi lehetőség a következő ábrán látható.

<PDP_type> Packet data protocol type. A string parameter which specifies the type of packet data protocol.
"IP" IPv4
"PPP"
"IPV6"
"IPV4V6"

Az utolsó paraméterében megadható kifejezés pedig az APN neve, emelyhez csatlakozni szeretnénk. Ezt fontos, hogy szöveges formátumban adjuk meg.

<APN> Access point name. A string parameter that is a logical name used to select the GGSN or the external packet data network. If the value is null or omitted, then the subscription value will be requested.

7. AT+CFUN=1

Ez a parancs felelős a modem bekapcsolására alapszinten, amennyiben ennek a parancsnak a paramétere 0-ra van állítva úgy a modem nem funkcionál. A parancs lehetőséget ad arra is, hogy segítségével újra indíthassuk a modemet, ilyenkor a „AT+CFUN=1,1” paramétert kell megadnunk neki. A parancs nagyon jó lehetőséget ad számunkra hibakeresésre is, ugyanis amennyiben ez a funkció nincsen 1-esre állítva, akkor nagyon sok problémát okozhat. Elsőnek tehát ezt a beállítást kell leellenőrizni, amennyiben hibát észlelünk. A parancs képes ERROR üzenetet is küldeni, ez akkor fordul elő, amennyiben nincsen a modembe behelyezve megfelelően a SIM kártya.

Parameter

<fun>	0	Minimum functionality
	1	Full functionality (Default)
	4	Disable the ME from both transmitting and receiving RF signals
<rst>	0	Do not reset the ME before setting it to **<fun>** functionality level. This is the default setting when **<rst>** is not given.
	1	Reset the ME. The device is fully functional after the reset. This value is available only for **<fun>**=1.

8. AT+COPS=1,2,"21630",8

Ez a parancs felelős, hogy a megfelelő mobilszolgáltatón keresztül csatlakozhassunk a Cat-M1 hálózatra. A paraméterlistáján sorrendben végig haladva, az első szám jelöli, hogy a mobilhálózat üzemeltetője elérhető. A második szám jelenti, hogy a szolgáltatót manuálisan fogjuk kiválasztani és ennek az azonosítóját mi adjuk majd meg a következő paraméterben. A harmadik és egyben legfontosabb paraméter tartalmazza a Service Provider azonosítására használt számsort. A számsornak két része van, az egyik az MCC (Mobile Country Codes) a másik pedig az MNC (Mobile Network Code). Az MCC felelős azért, hogy a mobilhálózatot be tudják azonosítani az országot, amelyhez tartozik a SIM kártya. Az MNC azt mutatja meg, hogy az országon belül melyik Service Provider hálózatát akarjuk használni. Az MCC Magyarországot jelölti (216) az MNC pedig a Magyar Telekom Nyrt. Vállalatot (30). Az MNC megegyezik azzal a számmal, amelyet tárcsázáskor is megadunk a hívószám elején.

A parancs utolsó paramétere kijelöli, hogy a Cat-M1 technológiát szeretnénk használni ezzel a végponttal. Ez a parancs nagyon sok információt tartalmaz a hálózatunkról és egyben az egyik legfontosabb része is a kapcsolódási fázisnak.

Example

```
AT+COPS=?                                   //List all current network operators
+COPS: +COPS: (3,"CHN-UNICOM","UNICOM","46001",0),(3,"CHINA
MOBILE","CMCC","46000",0),(1,"CHN-CT","CT","46011",9),,(0,1,2,3,4),(0,1,2)
```

Amennyiben eddig a pontig sikerrel eljutottunk, azaz minden parancsra „OK” választ adott a modemünk, akkor sikerült konfigurálnunk minden szükséges tényezőt ahhoz, hogy a

végpontunk csatlakozzunk a Cat-M1 hálózatra. Következésképpen van lehetőségünk forgalmat lebonyolítani az eszközünk segítségével. Nyomon tudjuk követni hálózati oldalról mennyi adatot forgalmaz, lekérdezhetjük az IP címét, távolról el fogjuk tudni érni az esetleges konfigurációk végett.

A következő lépésben azokat a hálózati paramétereket fogjuk beállítani, amelyek lehetővé teszik, hogy eszközünkkel adatforgalmat generálhassunk. A keretrendszerünk célja, hogy a szenzoradatot kinyerve, azt becsomagolva egy UDP datagram-ba kiküldhessük egy online platformra, ahol megjeleníthetjük az értékeit. Ennek előfeltétele, hogy a végpontunknak legyen IP címe, el tudjon érni külső eszközöket is, azaz kilásson az alhálózatából. Ennek feltétele a megfelelő átjáró (default gateway) felkonfigurálása. Amennyiben ezek a feltételek adottak, úgy egy csatornát (socketet) kell nyitni, amin keresztül a két végpont adatot tud küldeni egymásnak. Ehhez szükségünk van a két végpont IP címére és a portokra, amiket a kommunikáció idejére használatba vehetünk.

5.4.c A hálózati paraméterek vizsgálata

9. AT+CGPADDR

A fenti parancs segítségével kérdezhetjük le a modemünk IP címét, illetve győződhetünk meg arról, hogy nincsen neki.

AT+CGPADDR Show PDP Address	
Test Command **AT+CGPADDR=?**	Response **+CGPADDR:** (list of defined **<cid>**s) **OK**
Write Command **AT+CGPADDR[=<cid>[,<cid>[,...]]]**	Response **+CGPADDR: <cid>,<PDP_addr>** **[+CGPADDR: <cid>,<PDP_addr>[...]]** **OK** **ERROR**
Maximum Response Time	300ms
Reference 3GPP TS 27.007	

A válaszban egyértelműen megjelenik az eszközünk IP címe:

```
AT+CGPADDR=1                    //Show the PDP address
+CGPADDR: 1,“10.76.51.180”

OK
```

A modemünk IP címét is lekérdezve már minden olyan információval rendelkezünk, amely szükséges ahhoz, hogy adatot küldhessünk ki a Cat-M1mobilhálózaton keresztül. A csomagok küldéséhez szükségünk van egy hasznos adatra, amelyet a szenzorunk biztosít majd számunkra. A következő lépésekben, a socket nyitását fogom leírni és annak felparaméterezését a megfelelő protokollok használatának érdekében. A használt protokollok, amelyekkel a kommunikációt végezzük majd az UDP és a TCP lesz.

6. Firmware update over the air

A keretrendszerünkben az egyik legfontosabb elem a modem lesz, a mikroprocesszorában futó szoftver biztosítja számunkra a megfelelő működést. Amennyiben ez a szoftver valamilyen oknál fogva hibás úgy az IoT keretrendszerünkben is hibát fog okozni a működése. Fontos tehát, hogy a modemünket szoftveresen karbantartsuk, azaz a rajta található firmwaret updateljük. Ennek következtében a már letelepített keretrendszerekben is tudunk majd olyan változtatásokat tenni, amelyek le tudják követni a legfrissebb és leghatékonyabb kommunikációs formákat. A frissítés során kerülhetnek a modemen futó szoftverbe olyan újítások, amelyek hatására több protokollt is tud támogatni a rendszer és ezek a protokollok jobban menedzselhetővé teszik az adatküldéseket.

A Quectel BG96 modemünket több féle módon is frissíthetjük. Az egyik lehetőség, hogy manuálisan egy program segítségével feltöltjük rá a legfrissebb szoftververziót. Ezt a megoldást akkor tanácsos választani, amennyiben kevés eszközt használunk, esetleg csak tesztelésre. Amennyiben nagy számú, akár több ezer eszközünk is van, akkor ez a megoldás sajnos nem a leghatékonyabb. Ilyen esetekben használható az a metódus, hogy a modemünket távolról konfigurálva képesek vagyunk frissíteni a szoftverét. Ezt a műveletet hívják „firmware update over the air”-nek. A működését egy ábrán bemutatva:

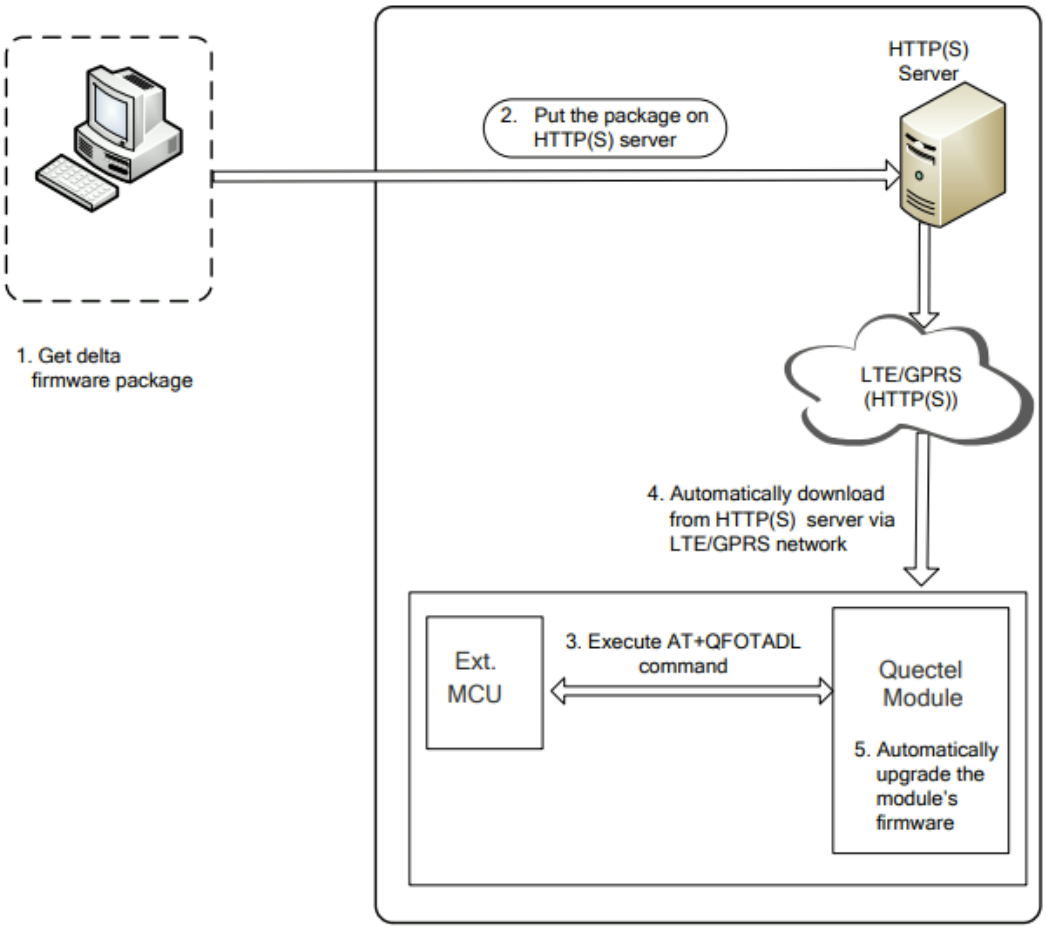


Figure 1: Firmware Upgrade Procedure via DFOTA

Az ábrán látható dolgok szerint 3 lépést kell megtenni a frissítés érdekében:

1. Legfrisseb szoftver verzió beszerzése
2. A verzió feltöltése egy http szerverre
3. A szerver megadása AT parancsban

A szoftver aktuális legfrissebb verzióját mindig a gyártótól kell igényelni. Mivel nagyon sok fejlődés várható ebben az iparágban elképzelhető és a tapasztalatok is ezt mutatják, hogy ezek a frissítések akár havi szinten is kijöhetnek egy-egy modemtípusra. A beszerzett firmware filet ezek után fel kell tölteni egy megadott http szerverre, ezt a szervert üzemeltetheti a gyártó. A gyártó álltal üzemeltetett szerver a legbiztonságosabb, így elkerüljük a biztonság kockázatokat, hogy esetleg felülírják a firmware filet és nem kívánt működésre bírják ezzel a modemünket. A biztonságos firmware update legjobb módja tehát a gyártón keresztül történik. Miután elhelyeztük a legfrissebb szoftverünket a szerveren, a következő lépés, hogy a modemünket is elvezessük erre a szerverre. Ezt AT parancsok segítségével tudjuk elérni, az egyik parancs paraméterében meg tudjuk adni a szerver URL vagy IP címét, amiről le tudja tölteni magára az általunk kiválasztott szoftvert.

A fimrware frissítéséhez a modem szemszögéből a következő parancsokat igényli:

1. Meg kell győződni, hogy a modemünk csatlakozva van e a Cat-M1 hálózatra.

```
AT+CGDCONT?                                    //Query APN setting
+CGDCONT: 1,"IPV4V6","Telstra.internet","0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0",0,0,0,0

OK
```

Amennyiben a parancsunk paraméterei megfelelnek az elvártnak úgy léphetünk a következő parancsokra.

2. További hálózatparmétereket kell lekérdeznünk, hogy az esetleges hibákat még a firmware frissítés előtt kezelni tudjuk.

```
AT+CSQ;+CREG?;+CGREG?;+COPS?          //Query network status
+CSQ: 26,99

+CREG: 0,1

+CGREG: 0,1

+COPS: 0,0,"Telstra Mobile Telstra",7

OK
```

Az első parancs „**AT+CSQ**” információt ad, hogy milyen térerősséget lát a modemünk. Amennyiben nem elegendő nem képes kommunikálni a hálózattal, ez az érték függhet környezeti hatásoktól, árnyékolástól, illetve a hálózati kommunikáció kódolásától is. Mivel a CAT-M1 IoT felhasználásra lett kifejlesztve, ezálltal olyan igényeket is ki kell szolgálni, ahol a hálózati jel

gyenge, mivel a szenzor esetleg a föld alatt található, vagy be van építve egy épület falába. A kommunikációt és a hálózatra való csatlakozást emiatt nagyon alacsony térerővel is képes lebonyolítani.

3. A harmadik lépésben adjuk meg a http szerverünk elérési útvonalát egy AT parancsba ágyazva.

```
//Execute AT+QFOTADL command to enable automatic firmware upgrade via DFOTA, and then the module will start to download the delta package and upgrade firmware automatically.

AT+QFOTADL="https://www.quectel.com:100/update.zip"
OK
+QIND: "FOTA","HTTPTART"
+QIND: "FOTA","HTTPEND",0          //Finish downloading the package from HTTPS server.
+QIND: "FOTA","START"
+QIND: "FOTA","UPDATING", 1%
+QIND: "FOTA","UPDATING", 2%
...
+QIND: "FOTA","UPDATING", 100%
+QIND: "FOTA","RESTORE", 1%
+QIND: "FOTA","RESTORE", 2%
...
```

Ez a parancs automatikusan letölti a szerverről a megadott firmwaret, ennek a folyamatáról állapotinformációkat is közöl, hogy éppen mekkora feldolgozottságnál jár a folyamat. A letöltés után fel is telepíti a modemre az új firmwaret.

7. A vezérlőn futó szofver

A következő pontban a vezérlőn (Raspberry Pi) futó szoftvert fogom bemutatni. A szofver feladata, hogy kinyerje az adatot a vezérlőre csatlakozott szenzorból, átformázza azt, hogy használható formátumú legyen az online platformra való kiküldésre. A program által készített adatforma már nem fog megváltozni, így annak kialakítása fontos pontja a keretrendszernek. Ez a program végzi el az összes konfigurációs feladatot, amely alapán szerenénk, hogy a modemünk működőképes legyen. Vezéreli a soros portját, amelyen keresztül kommunikál a modemmel, és adatot gyűjt a szenzortól.

7.1 A python nyelv bemutatása

A python függvények hasonlóan a C/C++ nyelvekhez NÉV+(PARAMÉTER) szintaktikával hívódnak meg. A python nyelv manapság az egyik legnagyobb fejlődéssel bíró programnyelv. Népszerűségét egyszerű olvashatóságának és könnyű kezelésének köszönheti, valamint széleskörű felhasználhatóságának is. Magában a függvényben fontos „includeolni” a megfelelő könyvtárakat, ezek nagyban segítik a feladat kivietlezését. Az includeolást követően, történik a scripteknek a megírása, ami a tényleges működést idézi elő. Változókat hozhatunk létre, ciklusokat indíthatunk. Linux rendszereken való python scriptek futtatása nagy rugalmasságot ad a programozónak. Bármilyen szövegszerkesztőt használunk (pl.: „nano”), már létre is hozhatunk egy python scripteket tartalmazó fájlt, figyelve arra, hogy a kiterjesztése *.py legyen. Ebben a fájlban megírhatunk olyan kódsorokat, amelyeket a command line ablakban futtatva előhívhatjuk annak működését. Futtatásukat a „sudo

python(3) fajlnev.py” paranccsal tudjuk elérni. Feladatom elvégzéséhez ezek a tulajdonságok nagy segítséget fognak nyújtani, mivel az interneten széles körben található jól leírt specifikáció az adott python könyvtárakhoz.

7.2 A vezérlő program bemutatása

A programkód bemutatását részekre fogom osztani, mégpedig a végrehajtódásuk szerinti sorrendben, a kód írását és az olvashatóságát könnyítve függvényekbe csomagoltam az parancsokat, így a fő programban csak a függvények hívását kell lebonyolítani a működés szempontjából. A tervezett bemutatása ezen függvényekkel fog kezdődni és a legvégén, a tényleges meghívásukat azoknak, ami végrehajtja azoknak a feladatát. A vezérlőn két program fog futni, az egyik, amely folyamatosan szolgáltatja a szenzorból kinyert adatot és a másik, amelyik feldolgozza azt és kiküldi a modem számára, hogy továbbítsa azt az online platformra. Először az adat kinyerését végző szoftvert fogom bemutatni.

7.2.a A szenzoradat kinyerése szofveres

A python program írásakor is az első és legfontosabb feladat a használni kívánt függvénykönyvtárak importálása. Amennyiben a kívánt könyvtár nincsen alapértelmezettként benne a telepített szoftverben, úgy azt külön le kell tölteni. Ezt a linuxos környezetben a legegyszerűbben a „pip” program segítségével tehetjük meg. A programot is telepíteni kell, amennyiben nincsen rajta a vezérlőnkön. A telepítéshez szükséges parancs a „sudo apt-get install python-pip”. Ezek után kikeresve a számunkra fontos könyvtárat már csak a következő parancsot kell kiadnunk, hogy elérhetővé váljon számunkra is a könyvtár: „sudo pip install könyvtár neve”. Az így feltelepített könyvtárat már importálatjuk a programkódunkba, használhatjuk beépített függvényeit, amelyek segítségével egyszerűbb és olvasható lesz a kódunk.

A szenzoradatot kinyerő szoftverben használt könyvtár a következő lesz:

```
import RPi.GPIO as GPIO

import time
```

Az első könyvtár, amelyet importálunk a Raspberry GPIO pineket menedzselni tudó függvényeket tartalmazza. Ennek a programnak a segítségével tudjuk kiolvasni az adatokat, amelyeket a szenzorunk kiüld. Vele tudjuk elérni azt, hogy a programunk futása után a pinek védelme érdekében, törölje az összes beérkező adatot, így csökkentve a meghibásodás kockázatát a vezérlőnknek.

A „time” programcsalád segít abban, hogy a modemünket ne terheljük le a nagy mennyiség adattal, amelyet egyidejűleg adnánk ki. Ezért a függvények meghívásával szüneteket iktathatunk be a parancsok között, időt adva a modemnek a feldolgozásra. Ez különösen fontos a rádiós interfacet konfiguráló AT parancsok esetében, hiszen a modem válasza csak akkor fog megérkezni, amint megtalálta a megfelelő hálózatot. Amennyiben a válasz előtt már új parancsot küldenénk a modemnek, az megzavarhatja a működés és nem kívánt hibákat okozna a keretrendszerünknek. A könyvtár funkciói segítenek továbbá a szenzoradat megfelelő kinyerésére, adott időközönként.

```
f = open('tmp.csv','a')

GPIO.setmode(GPIO.BOARD)

inpt = 13

GPIO.setup(inpt,GPIO.IN)

constant = 0.1006

time_new = 0.0

global rate_cnt, tot_cnt

rate_cnt = 0

tot_cnt = 0
```

Az adatok kinyeréséhez létre kell hoznunk különböző változókat. Ezekben a változókat fogjuk módosítani, hogy a végeredményben a megfelelő formátumú adatot láthassuk viszont. A programrészlet elején megnyitjuk azt a fájlt („`tmp.csv`”), amelyben tárolni fogjuk ideiglenesen a kinyert szenzor adatokat. A kinyert adatok a fogyasztás mértéke egy percre vetítve, illetve az összes fogyasztás. Be kell állítani a GPIO pineket „`BOARD`” módba, hogy a számuk alapján meghatározhatók legyenek, mivel a pinekre illesztettük a szenzorunkat. Az „`inpt = 13`” változó inicializálásával beállítottam, hogy a 13-as sorszámú GPIO pinen fog érkezni az adat, a szenzorunkból. A szenzor sárga vezetékét ezek után rákötöm a 13-as pinre, hogy fizikailag is megvalósuljon az összekötődés. A „`GPIO.setup(inpt,GPIO.IN)`” sor megadja, hogy a beállított pin input adatot szolgáltat majd a rendszerünknek, nem pedig output jelet. A „`constant`” változót a szenzorunk kalibrációja érdekében hozzuk létre, ezzel fogjuk ugyanis beállítani, hogy a kapott négyszögjelből majd tényleges fogyasztás mértékegységű adatot tudjunk készíteni, azálltal, hogy összeszorozzuk a megmért 1-es impulzusok számát ezzel a számmal. A „`time_new`” változót a ciklikusság érdekében hozzuk létre, mivel a percenkénti vízfogyasztást is mérni szeretnénk a rendszerünkkel. A látrehozott utolsó két változó segítségével az impulzusokat fogjuk mérni, amelyek beérkeznek a GPIO pinekre, ezeknek az értékét fogjuk növelni, amennyiben az input pinre impulzus érkezik. Azért kell kettő darab változó, mert két a két mért adat egyikénél a az összes impulzusra szükségünk van, míg a percenkénti vízmennyiség mérésénél egy perc letetle után kik kell nulláznunk és újrakezdenünk a számolást.

A következő programrészben az impulzusok számolását jelző változókat növelő függvényt fogom bemutatni.

```
def Pulse_cnt(inpt_pin):
      global rate_cnt, tot_cnt
      rate_cnt += 1
      tot_cnt += 1
```

Ez a függvény felelős azért, hogy a két változó értékét adott pillanatban növelje egyesével.

```
GPIO.add_event_detect(inpt,GPIO.FALLING,callback=Pulse_cnt,bouncetim
e=10)
```

```
rpt_int = 60

print('Reports every ', rpt_int, ' seconds')

print('Control C to exit')
```

A „GPIO.add_event_detect” függvényével képesek vagyunk a jelváltozásokat figyelembe venni. Amennyiben a paraméterében megadott input pinen impulzust érzékel, úgy meghívja a „Pulse_cnt” függvényt, amelyet a korábbiakban megírtunk, és elkezdni növelni a globális impulzusszámláló változóink értékét. Az „rpt_int” változóban megadjuk azt az értéket, amilyen időközönként szeretnénk a vízátfolyást mérni. A keretrendszer percenkénti vizsgálatot fog végezni, így ezt a változót 60-as értékre inicializáltuk.

```
def sensor():
      while True:
            time_new = time.time() + rpt_int
            rate_cnt = 0
            while time.time() <= time_new:
                  try:
                        None
                        print(GPIO.input(inpt), end='')
                  except KeyboardInterrupt:
                        print('\nCTRL C - Exiting nicely')
                        GPIO.cleanup()
            liter_per_min = round(rate_cnt * constant,2)
            total_liter = round(tot_cnt * constant,1)
            print(liter_per_min)
            print(total_liter)
            f.write(str(liter_per_min) + ' ')
            f.write(str(total_liter) + '\n')
            f.flush()
```

A szenzor adat kinyerését végző program fő része a fent ábrázolt függvény. Ez a függvény felelős, hogy a beérkező impulzusokat feldolgozza, átkonvertálja megfelelő formátumba, kiírja őket párosával a megadott fájlba, valamint a GPIO pinek védelme érdekében a kilépéskor elvégzendő adattisztítást is elvégezze. A függvény fő része egy végtelen ciklus, amely állandóan figyelemmel kíséri a GPIO pinekre érkező impulzusokat. Ezeket az impulzusokat a már meglévő változókban eltárolja (rate_cnt, tot_cnt) és beszorozva

ezeket a számokat a kalibráláshoz használt konstans változóval előkészíti a két hasznos adatot a fájlba írásra. A függvény utolsó parancsa „`f.flush()`”, nagyon fontos a keretrendszer működésének szempontjából. A parancs feladata, hogy a kiírandó adatot átmásolja a program bufferéből az operációs rendszer bufferébe. Ennek a haszna az, hogy amennyiben egy másik program is olvassa az eredményeket tartalmazó fájlt, úgy képes arra, hogy azonnal hozzájusson a legfrissebb adathoz, mivel azt az operációs rendszer írja bele a fájlba, nem kell várni a másik programra, hogy elvégezze az esetleges feladatait. A vezérlőn futó másik szoftver olvasni fogja az adatot ebből a fájlból, ezt nem tudná megtenni, amennyiben ezt az adatot nem adjuk ki.

A programunk működéséhez már csak a függvényt kell meghívnunk a „`sensor()`” szintaktikával. Ezt követően elkezdődik a folyamat, amelynek során kinyerjük az adatot a szenzorból, feldolgozzuk azt, átalakítjuk hasznos adattá, és el is tároljuk azt egy fájlban. A következő pontba a vezérlőn futó másik program kerül bemutatásra. Ez a program felelős a soros port vezérléséért, AT parancsok kiadásáért, valamint az online platformra kiküldendő csomag kódolásáért.

7.2.b A kinyert adat importálása a fájlból

Az általam használt könyvtárak a következők lesznek:

```
import sys

import glob

import serial

import time
```

A programom működése szempontjából fontos, hogy a „sys könyvtárat, felhasználjam. Ez a könyvtár alapértelmezetten megtalálható a python szoftverben így nem kell külön telepíteni azt. Segítségével a program futását tudjuk menedzselni, amennyiben ki akarunk lépni a programból, úgy azt a függvényei álltal el tudjuk érni. A függvényei segítségével command lineban megadott paramétereket is tudunk beolvasni és azokkal dolgozni. Nagyon hasznos könyvtár, amelyet szinten mindenhol használnak programfejlesztők.

A következő könyvtár a „glob”. A keretrendszer esetében ennek a könyvtárcsaládnak a segítségével tudunk olyan információkat megtudni, amelyek szükségesek ahhoz, hogy megállapítsuk a használni kívánt soros portot a kommunikációra. Ez a programcsalád képes feltérképezni és kiolvasni beépített függvényeivel olyan mappaszerkezeteket, amelyek alapján információt gyűjthetünk a vezérlő soros portjainak állapotáról. A linux rendszerünk tulajdonsága, hogy amennyiben új soros port nyitódik meg az eszközén, azt egy új alkönyvtár létrehozást vonzza ma után. Annak feltérképezése tehát információt ad nekünk a portok állapotáról.

A „serial” könyvtár az egyik legfontosabb könyvtár, amit használ a program a működése során. Ennek a könyvtárcsaládnak a függvényei végzik el a soros port vezérlését, beállítását. A soros portot megfeleltetjük egy objektumnak, amelyet aztán a könyvtár beépített függvényeivel beállítunk a megfelelő paraméterekre. Ennek a programnak a segítségével tudunk a soros porton keresztül kommunikálni a modemünkkel, adatot kiküldeni rá és a válaszokat megjeleníteni.

Az előző pontban bemutatott program előállítja a hasznos adatot, de ezt az adatot még át kell vinni abba a programrészbe, ahol a tényleges felhasználása fog megtörténni. Erre a feladatra a következő függvény lesz képes.

```
def read_data():
        with open('tmp.csv','r') as f:
                lastline = ""
                for line in f:
                        lastline = line
                perLiter = lastline[:3]
                fullLiter = lastline[:4]
                return perLiter, fullLiter
```

A függvény feladata, hogy megnyissa a „tmp.csv” fájlt, read-only módban, tehát csak olvasni tudjuk, módosítani nem. A fájl szóközzel ellátva két értéket tárol. Az egyik a fogyasztás literre vett értéke, a másik pedig az összes fogyasztás. Mivel a fájlba folyamatosan íródnak bele újabb és újabb sorok, a szenzorból kinyert adatokkal, így mindig a legutolsó sor tartalmazza a legfrissebb információt a fogyasztásunkról. Az utolsó sor megtalálása után (lastline) már csak fel kell darabolni két részre, hogy a két hasznos adatot el tudjuk raktározni két változóba a könnyebb felhasználás érdekében. Amint létrehoztuk a két változót és beletöltöttük az értékeket, vissza is térünk velük.

7.2.c Az aktív soros portok

A vezérlőn több soros port is megtalálható, de ezek közül nem mindegyik alkalmas arra, hogy kommunikációt folytassunk velük a modemmel. Egy specifikus port képes csak arra, hogy AT parancsokat küldhessünk ki rajta, ennek megtalálása viszont nem mindig triviális. Az a megoldás nem vezet sikerre, hogy az összes portot végig próbáljuk, hátha az lesz a megfelelő. A legjobb megoldás erre, ha feltérképezzük a portokat, amelyek elérhetőek az operációs rendszerünkben és azoknak a csoportját szűkítjük bizonyos tulajdonságok keresésével, amíg meg nem találjuk a számunkra fontos portot. Az első feladat tehát, hogy a portokat kilistázzuk, amelyek elérhetőek a vezérlőnkön keresztül, ezt a feladatot végzi el az alábbi függvény.

```
def serial_ports():

     if sys.platform.startswith('win'):

          ports = ['COM%s' % (i + 1) for i in range(256)]

     elif sys.platform.startswith('linux') or
sys.platform.startswith('cygwin'):

     # this excludes your current terminal "/dev/tty"

          ports = glob.glob('/dev/tty[A-Za-z]*')

     elif sys.platform.startswith('darwin'):

          ports = glob.glob('/dev/tty.*')

     else:
```

```
            raise EnvironmentError('Unsupported platform')
      resoult = []
      for portok in ports:
            try:
                  device = 'ttyUSB'
                  str_serial =
str(serial.Serial(portok,rtscts=True,dsrdtr=True))
                  if
serial.Serial(portok,rtscts=True,dsrdtr=True).isOpen() == True and
str_serial.find(device) > 0:
                        resoult.append(portok)

      serial.Serial(portok,rtscts=True,dsrdtr=True).close()
            except (OSError, serial.SerialException):
                  continue
      return resoult
```

A vezérlő eszközünkön futó operációs rendszer különböző lehet. A mi esetünkben ez a rendszer egy linux alapú konstrukció. A függvény ezt a tulajdonságot szűri ki legelőször, hiszen a linux rendszerekben az új soros portot egy új mappa megjelenésével lehet automatizálva detektálni. Az operációs rendszerünk (linux) mivoltáról a következő parancs kiadásával tudunk információkat begyűjteni „`sys.platform.startswith('linux')`” . Amennyiben ez a feltétel teljesül, úgy a függvény tudni fogja, hogy az a rendszer, amelyen ő fut, az linux alapokon nyugszik. Ennek az információnak a tudatában elkezdi feltérképezni az összes olyan portot képviselő mappát, amely potenciális AT portként funkcionálhat majd. A mappaszerkezetek átkutatásához a „`glob`” könyvtár függvényeit használja, ahogy ez a „`ports = glob.glob('/dev/tty[A-Za-z]*')`” parancsnál is látszik. A linux rendszereken létrejövő portmappák nevei mindig a „tty” azonosítóval kezdődnek. Ez az azonosító a korai linux rendszereknél jött létre és a „TeleTYpewriter” rövidítésként értelmezhető. Ezek a mappák, amelyek egy virtuális soros portként is kezelhetőek el lesznek raktározva egy tömbben, amelyet a „glob” hoz létre. Ezeket a mappákat utána még egy körben átszűri a függvény, az alapján, hogy talált e a nevükben „USB” kulcszót. Mivel a modemünk egy USB porton keresztül van fizikailag összekötve a vezérlőnkkel, így a soros portok közül egy „ttyUSBx” mappán keresztül tudjuk elérni azt. Az „x” egy számot jelöl, mivel modem fajtától függően akár több virtuális portot is létrehozhat, amikor csatlakozik a vezérlőhöz. Az eredményt, amely most már csak azokat a mappákat tartalmazza, amelyek USB virtuális portként foghatók fel egy tömbben helyezi el. Ezzel a tömbbel is tér vissza, hogy a későbbekben az eredményét fel lehessen használni.

7.2.d A használni kívánt port kiválasztása

Az előző függvény feltérképezte és összegyűjtötte a számunkra fontos portokat. Ezeket a portokat viszont még meg kell egyszer szűrni, mert nem mindegyik funkcionálhat AT portként. Azt a tulajdonságát, hogy tud e AT parancsokat küldeni és a válaszukat fogani egyszerű teszteléssel éri el a függvény.

```
def find_ports_for_AT(portlist):
	msg1 = "AT\r"
	msg2 = "AT+CFUN?\r"
	ok = "OK"
	AT_port = []
	for i in portlist:
		ser = 
serial.Serial(i,baudrate=115200,timeout=1,xonxoff=True,rtscts=True,d
srdtr=True)
		ser.write(msg1.encode())
		ser.write(msg2.encode())
		if str(ser.read(64)).find(ok) > 0:
			AT_port.append(i)
	return AT_port
```

Minden megkapott portra kiküld kettő-kettő AT parancsot, és a válaszokból megállapítja, hogy az a port alkalmas e a kommunikációra vagy nem. A két választott parancs az „AT” ás az „AT+CFUN”. Mindkét parancs választ is vár a modemtől így ezeket kiadva megbizonyosodhatunk a soros portunk helyes működéséről. Ezt a metódust megismételi az összes portra, amelyket az előző függvény előállított a számára. Azt a portot vagy portokat, amelyek megfeleltek az elvárásnak, tehát fogadták az AT parancsot és választ is tudott a modem küldeni rajtuk keresztül elhelyezzük egy tömbben, és visszatérünk az elemeivel.

7.2.e Csatlakozás a hálózatra szoftveresen

A programunk egyik legfontosabb függvényének bemutatása lesz leírva ebben a pontban. A metódus feladata, hogy a paraméterében megadott AT porton keresztül kommunikáljon a modemmel. Kiadja a megfelelő parancsokat és a válaszokat elraktározza egy tömbben, majd a függvény végére érve visszatérjen a válaszokkal. A program ellenőrzi, hogy a paraméterben megadott port tényleg nyitva van e. Ezt a tulajdonságát a portnak a „`serial.Serial(port,rtscts=True,dsrdtr=True).isOpen()`” sor segíségével tudja megvizsgálni. Amennyiben erre a parancsra a válasz az, hogy „`True`”, akkor a port tényleg aktív és lehet használni.

```
def attach_to_network(port):
	if serial.Serial(port,rtscts=True,dsrdtr=True).isOpen() == 
True:
```

```
            out = []
            try:
                  ser =
serial.Serial(port,baudrate=115200,rtscts=True,dsrdtr=True)
                  ser.write(('AT+QCFG="nbsibscramble",0\r').encode())
                  time.sleep(1)
                  while ser.inWaiting() > 0:
                        out.append(ser.read(ser.inWaiting()))

      ser.write(('AT+CPSMS=0,,,"00000100","00001111"\r').encode())
                  time.sleep(1)
                  while ser.inWaiting() > 0:
                        out.append(ser.read(ser.inWaiting()))

      ser.write(('AT+QCFG="band",0,80000,80000,1\r').encode())
                  time.sleep(1)
                  while ser.inWaiting() > 0:
                        out.append(ser.read(ser.inWaiting()))
      ser.write(('AT+QCFG="nwscanmode",1\r').encode())
                  time.sleep(1)
                  while ser.inWaiting() > 0:
                        out.append(ser.read(ser.inWaiting()))
      ser.write(('AT+QCFG="nwscanseq",020301\r').encode())
                  time.sleep(1)
                  while ser.inWaiting() > 0:
                        out.append(ser.read(ser.inWaiting()))
      ser.write(('AT+QCFG="iotopmode",1\r').encode())
                  time.sleep(1)
                  while ser.inWaiting() > 0:
                        out.append(ser.read(ser.inWaiting()))

      ser.write(('AT+QCFG="servicedomain",1,0\r').encode())
                  time.sleep(1)
```

```
            while ser.inWaiting() > 0:
                  out.append(ser.read(ser.inWaiting()))

      ser.write(('AT+CGDCONT=1,"IP","internet.telekom"\r').encode())
            time.sleep(1)
            while ser.inWaiting() > 0:
                  out.append(ser.read(ser.inWaiting()))
      ser.write(('AT+COPS=1,2,"21630",8\r').encode())
            time.sleep(1)
            while ser.inWaiting() > 0:
                  out.append(ser.read(ser.inWaiting()))
      ser.write(('AT+CGPADDR=1\r').encode())
            time.sleep(1)
            while ser.inWaiting() > 0:
                  out.append(ser.read(ser.inWaiting()))
            if ser.inWaiting() == 0:
                  return out
         except KeyboardInterrupt:
            None
```

A program azokat a parancsokat küldi ki a soros portján keresztül, amelyeket az 5.4.b pontban lettek összegyűjtve. Fontos tényezője a függvénynek, hogy a parancsok kiadását követően, kis időt hagy a modemnek a feldolgozásra. Ez nagyon fontos, főleg a hálózat detektálásánál, amikor a válasza a modemnek nem azonnal érkezik meg. Ez a kis várakozási idő kiszűri azon problémákat, amelyek abból adódhatnak, hogy egy parancs még nem került feldolgozásra a modemnél, de egy következő parancs már beérkezett. A program összegyűjti a modem válaszait egy tömbbe, mégpedig úgy, hogy a tömb elemeihez mindig hozzáadja azt a mennyiséget, amelyik éppen a soros port input bufferében találhatóak. Ezt a folyamatot szemlélteti a „`out.append(ser.read(ser.inWaiting()))`” parancssor. Ez a parancs hozzáadja a kimeneti tömbhöz a soros port inputján érkező modemválaszt.

7.2.f Adatküldés az online platformra

Az adatok kiküldése IP felett fog történni, a kipróbált és letesztelt protokoll a TCP lesz, amely megbízható kapcsolatot tart fent csomagküldés szempontjából a kliens és a szerver között. A szerver jelen esetben egy online platform lesz (ubidots.com), a kliens pedig a modem. A modemnek, amint csatlakozik a hálózatra lesz IP címe, azaz képes küldeni és fogadni IP csomagokat.

```
def send_data_tcp():
      data = craete_data_for_ubidots()
```

```
      if serial.Serial(port,rtscts=True,dsrdtr=True).isOpen() ==
True:
            out = []
            error = "ERROR"
            try:
                  ser.write(('AT+QIACT=1\r').encode())
                        time.sleep(2)
                  while ser.inWaiting > 0:
                        out.append(ser.read(ser.inWaiting()))
                  ser.write(('AT+QICLOSE=0\r').encode())
                  time.sleep(2)
                  while ser.inWaiting > 0:

out.append(ser.read(ser.inWaiting()))

      ser.write(('AT+QIOPEN=1,0,"TCP","translate.ubidots.com",9012\r'
).encode())
                  time.sleep(2)
                  while ser.inWaiting > 0:

out.append(ser.read(ser.inWaiting()))
                  ser.write(('AT+QISENDEX=0,' + '"' + data + '"' +
'\r').encode())
                  time.sleep(2)
                  while ser.inWaiting > 0:
                        out.append(ser.read(ser.inWaiting()))
                  ser.write(('AT+QIRD=0\r').encode())
                  time.sleep(2)
                  while ser.inWaiting > 0:
                        out.append(ser.read(ser.inWaiting()))
            except:
                  None
```

```
        if error in out[3]:
            return True
        else:
            return False
```

A kommunikáció egy socketen keresztül valósul meg. Ezt a socketet kell megfelelően paraméterezni, ahhoz, hogy a kívánt helyre el tudjuk küldeni az adatunkat. A socketnek meg kell adni, hogy mi a szerver IP címe, vagy domain neve, valamint azt, hogy a szerver melyik portján képes fogani a csomagokat. Ezeket a paramétereket állítja be a függvény a „`ser.write(('AT+QIOPEN=1,0,"TCP","translate.ubidots.com",9012\r').encode())`” programsorban. Az AT parancs megnyit egy socketet, aminek a sorszáma „0” lesz, kiválasztja a kommunikáció alapjául szolgáló protokollt a szerver IP címét, valamint a szerver portját. Amennyiben ez a socket index, amit kiválasztottunk (0) már foglalt, akkor a függvény bezárja azt és újat nyit a fenti paraméterek megadásával. A következő lépésben a megnyitott socketen kiküldjük az IP comagot, a szerver részére, ebben a csomagban található meg a hasznos adat, illetve az online platformon létrehozott felhasználói fiokot azonosító egyedi kulcs (token). Ezt a feladatot a „`ser.write(('AT+QISENDEX=0,' + '"' + data + '"' + '\r').encode())`” parancs hajtja végre. A TCP specifikációja alapján, a szerver tájékoztatja a klienst a fogadott adatok mennyiségéről válasz formájában. Ahhoz, hogy a fogadott adatokat meg tudjuk jeleníteni, a „`ser.write(('AT+QIRD=0\r').encode())`” parancsot kell kiadnunk. Ez a parancs megjeleníti a paraméterében (=0) megadott socketre érkező input adatokat.

7.2.g Az adat konvertálása

Fontos, hogy a hasznos adatunkat olyan formára konvertáljuk át, amit TCP csomagba ágyazva képes az online platform feldolgozni, illetve a modem is képes kezelni. Erre a feladatra fog megoldást találni a „`craete_data_for_ubidots()`” függvény. Ez a függvény meghívja az hasznos szenzoradatot kinyerő függvényt „`read_data()`” és összekapcsolja egy nagy hexadecimális sztringbe az adatot. Ebben az adatban benne van a két hasznos adat, amit a szenzor kinyer, a token, ami a felhasználót azonosítja az online platformon.

```
def craete_data_for_ubidots():
    data1, data2 = read_data()
    token = "BBFF-aStyrZ84Id5yovR5YpYs5kQo0UKfq5"
    tmp = "microchip/1.0|POST|" + str(token) + "|bg96=>masik:" +
str(data1) + ",egyik:" + str(data2) + "|end"
    to_hex = tmp.encode("utf-8").hex()
    return to_hex
```

7.2.h A fő program, függvények hívása

A program legvégén a szükséges függvények meghívása történik, ami a tényleges megvalósítást jelenti. Ezt a kódot futtatva, már tesztelni tudjuk a keretrendszerünk működését éles helyzetben is.

```
if __name__ == '__main__':
      portok = serial_ports()
      at_ports = find_ports_for_AT(portok)
      print(at_ports)
      AT_PORT = select_port_for_use(at_ports)
      print(AT_PORT)
      network = attach_to_network(AT_PORT)
      print(network)
      if send_data_tcp():
            print("Sikerült!")
```