Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

Méréstechnika és Információs Rendszerek Tanszék

Vezeték nélküli okosmérőóra-hálózat tervezése és szoftver implementációja

Készítette

Molnár Dániel

Konzulens

Dr. Orosz György

Molnár Károly

2016

MŰEGYETEM 1782

DIPLOMATERV-FELADAT

Molnár Dániel (YNV3MS)
szigorló villamosmérnök hallgató részére

Vezeték nélküli okosmérőóra-hálózat tervezése és szoftver implementációja

A dolgozat a vezeték nélküli szenzorhálózatok egy speciális alkalmazási területével foglalkozik. A meglévő okos mérőóra technológiákban központosított infrastruktúrát használnak. A robusztusság, illetve a lefedettségi terület növelése érdekében kézenfekvő megoldás lehet a vezeték nélküli hálózati megoldások használata.

Az okos mérőórák (SmartMeter) alkalmazásának egyeik lehetséges módja, ha az okos mérőórák rádiós kommunikációs kapcsolaton alapuló szenzorhálózatot alkotva működnek. Ennek elsődleges felhasználási területe a sok lakásos társasházak és tömbházak. A dolgozat célja egy mérőórákból kialakított kísérleti vezeték nélküli okosmérőóra-hálózat teljes körű szoftvertervezése és implementációja.

A kísérleti rendszer hardver elemeit adottnak tekintjük, az előírt rádiós kommunikációs szabvány a WMBUS. A kísérleti rendszer alapvetően két fő részből áll, a mérőórák mellé telepítendő kommunikációs modulokból és egy központi gateway egységből, amely a vezeték nélküli hálózat központi eleme, valamint ez biztosítja a kapcsolatot a külvilág (távoli szerver) felé. A gateway előírt operációs rendszere beágyazott Linux vagy Android.

A feladat a vezeték nélküli hálózat kommunikációjának megtervezése, a megfelelő protokollok kiválasztása, a kommunikációs modulok és a központi egység vezérlő szoftverének megtervezése és implementálása. Elvárás a titkosított adatkommunikáció, tehát autentikáció, adattitkosítás és kulcskezelés is szükséges.

A hallgató feladatának a következőkre kell kiterjednie:

- Tekintse át a szükséges tudományos irodalmat a vezeték nélküli hálózatok algoritmusai körében, és elemezze a lehetséges megoldásokat a konkrét alkalmazás szempontjai alapján!
- Tervezze meg a hálózati kommunikáció rétegeit, és a hálózati csomópontok működését!
- Tervezze meg és implementálja a kommunikációs modul beágyazott firmware-t!
- Tervezze meg és implementálja a gateway vezérlő szoftverét!
- Demonstrálja a működést!

Tanszéki konzulens: Orosz György, adjunktus
Külső konzulens: Molnár Károly, ProDSP Kft.

Budapest, 2016. március 18.

.....
Dr. Dabóczi Tamás
tanszékvezető

Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Méréstechnika és Információs Rendszerek Tanszék

1117 Budapest, Magyar Tudósok krt. 2. I. ép. I.E.444.
Telefon: 463-2057, Fax: 463-4112
http://www.mit.bme.hu • e-mail: mitadm@mit.bme.hu

TARTALOMJEGYZÉK

HALLGATÓI NYILATKOZAT

Alulírott Molnár Dániel, szigorló hallgató kijelentem, hogy ezt a diplomatervet nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy hitelesített felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Kelt: Budapest, 2016. 12. 18.

.....

Molnár Dániel

Kivonat

Jelen diplomaterv célja egy vezetéknélküli okosmérőóra-hálózat tervezésének és szoftver implementációjának bemutatása. Az elkészült rendszer alapot adhat háztartások energiafogyasztásának távoli mérésére, mely mind a szolgáltatónak, mind a felhasználóknak előnyöket teremthet. Ezt a mérőórák mellé telepített rádiós egységek, valamint egy adatkoncentrátor által létrehozott rádiós hálózat teszi lehetővé. Az adatkoncentrátor távoli eléréssel rendelkezik egy szerverhez, mely tárolja, valamint feldolgozza a mérési adatokat.

A létrehozott rádiós hálózat Wireless M-Bus fizikai réteggel rendelkezik mely egy európai szabvány kifejezetten okosmérőórák részére. A Wireless M-Bus szabvány nem specifikál hálózati réteget, így a feladat magába foglalja annak implementálását is. A hálózat a kommunikáció formája, kiterjedése, valamint a felhasználása szempontjából felfogható egy speciális vezetéknélküli szenzorhálózatként, így a tervezés során az ezen témakörben ismert útkeresési, valamint biztonsági eljárások kerülnek figyelembe vételre.

A hálózat tervezésekor a cél egy robosztus multi-hop hálózat létrehozása, amely kiküszöböli a beltéri rádiós kommunikáció rövid hatótávját, valamint lehetővé teszi a működést mérőórák esetleges meghibásodása esetén is. További cél a hálózat könnyű bővíthetősége. A titkosítás az alkalmazás szempontjából igen fontos, mivel a kommunikáció, valamint az adatok meghamisítása nagy károkat okozhat a szolgáltatóknak és a felhasználóknak is.

A dolgozat során definiálásra kerülnek az egyes rendszerelemek, melyekkel szemben támasztott hardveres, illetve szoftveres követelmények lefektetése után azok implementálásra is kerülnek.

Abstract

The goal of my thesis is the design of a network of smart-meters and the implementation of their software. The network can measure and store the consumption data of household electricity and enables further services through an extension with central server communication. The network that is created between the smart-meters uses Wireless M-Bus physical layer, which is a European standard designed specifically for smart-meters.

The standard does not specify the network layer, therefore designing it is part of the work of my thesis. A smart-meter network is similar to a wireless sensor network, therefore the solutions and algorithms used for wireless sensor networks in terms of routing, synchronization and encryption can be studied and applied here as well.

During the design of the network, the goal is to create a robust multi-hop network, which can eliminate the short range of indoor radio communication, as well as enables normal operation even if a smart-meter malfunctions or is not available. Another goal of the design is expandability. Encryption is an important aspect of the network, because falsified communication or data can cause major harm for both the provider and the end user.

In the thesis, the firmware of the network devices responsible for the operation of the network and the software running on the embedded Linux operating system is implemented. The latter software processes the messages transmitted on the network, it communicates with the central server through a GSM connection and it is also responsible for central data processing.

1. Bevezetés

A háztartási mérőórák a háztartásban fogyasztott villany, gáz, illetve vízfelhasználást mérik. A mérési adatok elsősorban a szolgáltatóknak biztosítanak visszajelzést az általuk nyújtott szolgáltatás felhasználásának mértékéről, mely alapján kiszabják a díjat. Emellett a mérési adatok egyéb hasznos információval is szolgálnak, mely alapján a szolgáltató optimálisan képes elosztani az erőforrásait. Ilyen lehet például a nagyobb fogyasztással bíró területek infrastruktúrájának fejlesztése. A mérési adatok a felhasználóknak is fontos visszajelzéssel szolgálnak, mely alapján pénzügyi vagy környezettudatossági szempontok alapján alakíthatják energiafelhasználási szokásaikat. A klasszikus mérőóra infrastruktúra adatgyűjtése a fizikai leolvasás miatt sok emberi erőforrást igényel, továbbá a visszajelzés gyakorisága sem sűrű, általában havonta történik meg. Ez nem ad lehetőséget gyors reakcióra sem a szolgáltatónak, sem a felhasználóknak. További, manapság elterjedt módszer a mérőóra állás felhasználó általi telefonon történő bemondása, mely a mérési-leolvasási hibák számát növeli.

Az okosmérőóra-hálózat klasszikus mérőórák kiegészítésével vagy lecserélésével, valamint a szükséges kommunikációs infrastruktúra kialakításával érhető el, amely lehet vezetékes vagy vezeték nélküli. A vezetékes kommunikáció használatát indokolhatja egy már meglévő csatorna használata, úgy mint a villamos távvezetéken történő adattovábbítás (*power line communication*). Az ilyen rendszerek nagyobb bonyolultságú és költségű infrastrukturális hátteret igényelnek, valamint az újrakonfigurálhatóság szempontjából sem előnyösek. Ezzel szemben vezeték a nélküli technológiák könnyű telepítést és konfigurálhatóságot tesznek lehetővé, így a tervezéskor az ISM-sávú rádiós technológiákra fogunk hagyatkozni. A vezetéknélküli technológiák velejárója a korlátozott hatótávolság, mely jelen alkalmazásban tovább romlik a felhasználásból származó fizikai adottságok miatt. Ilyen adottság az, hogy a mérőórák lakóházakon belül kerülnek elhelyezésre, mely családi házak esetén alapvetően nagyobb távolságokat jelent az egyes mérőórák között, illetve panel lakások esetén több falréteget (mely általában vasbeton szerkezetű) jelent, így csökkentve a rádió hatótávolságát.

Egy, a területen alkalmazott és elterjedt szabvány az MBUS, melynek vezeték nélküli variánsa az EN13757-4 kiadványban [1] szerepel. Ez az OSI személettel vizsgálva a fizikai, valamint az adatkapcsolati réteget definiálja, azonban a hálózati réteget nem.

Ezen dolgozat célja egy olyan vezetéknélküli WMBUS okosmérőóra-hálózat tervezése, mely lehetővé teszi a mérési adatok egy központi egység felé történő biztos továbbítását egy lokális hálózat felépítésével. A felépülő lokális hálózatnak könnyen telepíthetőnek, valamint robosztusnak kell lennie. A robosztusság alatt azt a tulajdonságát értjük a hálózatnak, mely szerint beavatkozás nélkül képes kezelni új mérőórák a hálózatba történő becsatlakozását, valamint azok esetleges meghibásodásait, így a hálózatból történő kiesésüket. A hálózat további követelménye az egyes egységek korlátolt hatótávolságának kiküszöbölése a mérőórák együttműködésének segítségével, annak érdekében, hogy a mérési adatokat eljuttassák a központi feldolgozóegységnek. A tervezendő hálózat adottságai, valamint a már ismertetett problémák sok hasonlóságot mutatnak a vezetéknélküli szenzorhálozatokhoz (*Wireless Sensor Networks*), azok egy speciális területeként tekinthetők, így a problémák vizsgálatakor az ott használatos eljárásokat és algoritmusokat fogjuk megvizsgálni. A különbség a klasszikus szenzorhálozatokhoz képest a node-ok alacsonyabb száma, a topológia viszonylagos statikussága, valamint az, hogy az elhelyezésből adódóan van lehetőség hálózati tápellátásra, így nem fontos szempont az alacsony fogyasztás.

A rádiós kommunikáció velejárója a viszonylag könnyű lehallgathatóság, ami nagy biztonsági kockázattal jár mind a szolgáltatónak, mind a felhasználóknak, ezért szükséges az adatok titkosítására is megoldást találnunk. Alapvetően két titkosítási módszerrel foglalkozunk, a szimmetrikus, valamint az aszimmetrikus kulcsú titkosítással. A megfelelő adattitkosítási eljárás tervezésekor figyelembe kell vennünk a titkosítási módszerek előnyeit és hátrányait, az optimális megoldást keresve.

A fejlesztés a ProDSP Technologies Kft. gondozásában történik, és hardver-, illetve szoftverfejlesztés is tartozik hozzá. A hardver fejlesztését Tóth Eszter végzi diplomaterv keretein belül [2], továbbá a hardveres egységek kiválasztását szoftveres szempontok figyelembevételével segítem.

A 2. fejezetben ismertetésre kerül a rendszer magas szintű specifikációja, részletezve a fő szempontokat, a 3. fejezetben bemutatok néhány potenciálisan megfelelő routing algoritmust az adattovábbító hálózat kiépítéséhez, a 4. fejezetben kitérünk a biztonság és titkosítás kérdésére, az 5. fejezetben röviden ismertetésre kerül a Wireless M-Bus vezetéknélküli mérőóraszabvány, a 6. fejezetben a rendszer, illetve a rendszerelemek tervezése és implementációja kerül részletes bemutatásra, míg a 7. fejezetben a rendszer működésének demonstrálására kerül sor.

2. Specifikáció

A feladat magába foglalja a rendszer teljes megtervezését, melyhez hozzátartozik a node-ok, valamint a központi feldolgozó egység követelményeinek specifikációja is.

A rendszer általános tulajdonságai a következők:

- Okosmérőórák mellé telepíthető kommunikációs egységek
- Robosztus vezetéknélküli (rádiós) hálózat
- Nagy számú mérőóra
- Titkosítás
- Egy gateway vezérlő egység a rádiós hálózat és a távoli szerver között
- Távoli adattárolás, feldolgozás, és megjelenítés

Az alábbi alfejezetekben a rendszer legfőbb jellemzői kerülnek részletes specifikációra.

2.1. Wireless M-Bus mérőórahálózat

Az M-Bus [1] az okosmérőóra technológiában széleskörűen alkalmazott európai szabvány, mely a kommunikáció fizikai, adatkapcsolati, valamint alkalmazási rétegeit határozza meg. Célja egy olyan kommunikációs protokoll definiálása, mely a mérőóra-hálózatok igényeit széleskörűen és egyszerűen kielégíti. A szempontok között a nagy számú eszköz összekapcsolása, a hálózat egyszerű bővítése, a robosztusság, az alacsony költség és fogyasztás, valamint megbízható működés szerepel. A szabvány tartalmaz vezetéknélküli alkalmazást is, melynek használata a feladat egyik specifikációs pontja. Használatát az indokolja, hogy a piacon megtalálható okosmérőórák és okosmérőóra-rendszerek széles körben ezt a szabványt támogatják.

A szabvány nem definiál hálózati réteget, így annak implementálása szabadon választható, mindössze a változó csomagméret támogatása követelmény.

2.2. Routing (útvonalválasztás)

A felépülő hálózat működése kiemelten fontos az okosmérőóra szolgáltatás minőségének garantálásához. A rendszerben továbbá meghibásodások esetén minimálisnak kell lennie a karbantartási és javítási időtartamnak. További szempont még az egyszerű telepíthetőség is. Ezen követelmények kerülnek alább bővebb kifejtésre.

A tervezendő rendszer a nagy számú felhasználó miatt megköveteli az olcsó és egyszerű telepíthetőséget. Ez azt jelenti, hogy a node-ok fizikai kitelepítése után a lehető legkevesebb emberi beavatkozással épüljön fel a routinghoz szükséges hálózat, valamint a rendelkezésre állás megkezdése is minél hamarabb elkezdődjön.

Mivel a kitelepítendő eszközök háztartásokhoz tartoznak, ezért azok nem ad-hoc módon kerülnek elhelyezésre, továbbá rendelkeznek egyedi azonosítókkal mely segítségével a node-ok adatai hozzárendelhetők a felhasználókhoz. Ez a telepítés során segítséget nyújthat különböző eljárásoknál, valamint felvezeti a hálózat következő követelményét is.

A rendszer egyedi felhasználók adatait dolgozza fel, így szükséges azok azonosítása, sok klasszikus szenzorhálózati alkalmazással szemben. Lehetőséget kell biztosítanunk továbbá az esetleges további funkcióbővítés során felmerülő, a hálózat elemeibe történő szolgáltató felőli beavatkozásra is, melynek csakugyan elengedhetetlen követelménye az egyedi címezhetőség. Ehhez szükséges a kétirányú (bidirekcionális) kommunikáció támogatása, mely egy újabb szempont az algoritmusok vizsgálatánál.

A hálózattal szemben támasztott egyik legfontosabb követelmény a robosztusság. Ez magában foglalja azt a tulajdonságot, hogy egyes node-ok a hálózatból történő kiesésekor is képes a hálózat a többi node adatait a központi feldolgozóegység felé közvetíteni, valamint támogatja új node-ok becsatlakozását is.

2.3. Titkosítás

A hálózatban alkalmazott rádiós kommunikáció velejárója a viszonylag könnyű lehallgathatóság, ami nagy biztonsági kockázattal jár mind a szolgáltatónak, mind a felhasználóknak.

A biztonsági kérdések első szintje a mérőóráknál illetve az okosmérőóra-rendszert megvalósító hardvernél kezdődik, a fizikai beavatkozások területén. A mérőórák érzékelői, számlálói mechanikai behatolással oly módon módosíthatók, hogy az a szolgáltatónak anyagi kárt okozzon, így a hálózati kommunikáció biztonsága érdekében előfeltétel a fizikai biztonság megléte. További lehetséges hardver támadás a rendszer elemeinek másolása. Ezen dolgozatnak nem célja a hardveres biztonsági problémák kezelése, csupán a kommunikáció titkosításával foglalkozik.

Titkosítatlan kommunikáció lehallgatása a támadóknak lehetőséget ad a rendszerben történő folyamatok feltérképezésére, illetve lehetőségük nyílik az üzenetek egy külső eszközzel történő replikálására valamint módosítására, például a küldött óraállás módosítása a valóságosnál kisebb értékre.

A dolgozat célja egy olyan titkosítási módszer implementálása, amely megvédi a rendszert a kommunikációt lehallgatóktól.

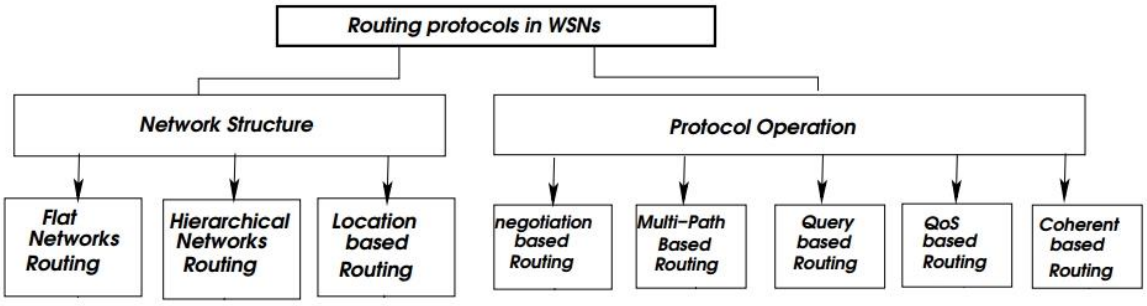
3. Routing (útvonalválasztó) algoritmusok

A vezetéknélküli szenzorhálózatok alap feladata fizikai vagy környezeti adatok mérése (hőmérséklet, nyomás stb.), és ezen mérési adatok továbbítása egy központi feldolgozó-egység felé. Az ilyen hálózatok általában nagyobb kiterjedésűek mint az egyes vezeték-nélküli eszközök hatótávolsága, így pont-pont összeköttetésre a szenzor node valamint a feldolgozó node között nincs lehetőség, ezért megoldandó probléma az adatok eljuttatása. Ezen problémakörrel foglalkoznak a routing algoritmusok. Az olyan hálózatokat, amik a pont-pont összeköttetés lehetőségének hiányát küszöbölik ki, multi-hop hálózatoknak nevezzük.

Mint az már a bevezetőben említésre került, az okosmérőóra-hálózatok a vezetéknélküli szenzorhálózatok egy speciális területeként tekinthetők a rádiós kommunikáció, a nagy szenzor számosság és kiterjedés, valamint a korlátozott hatótávolság miatt.

A routing algoritmusok vizsgálatakor az alkalmazásunk szempontjából több lényeges követelményt is teljesítenünk kell, melyek a 2.2. fejezetben kerültek ismertetésre.

A továbbiakban bemutatásra kerül több, a vezetéknélküli szenzorhálózatoknál alkalmazott routing algoritmus, annak érdekében, hogy egy, az alkalmazás szempontjából optimális megoldás kiválasztásra, implementációra vagy személyre szabásra kerülhessen. A routing algoritmusokat több szempontból, mint például a hálózat struktúrája vagy a protokoll működése szerint vizsgálhatjuk, illetve ezeken belül is tehetünk különbségeket köztük, melyet a 3.1. ábra szemléltet.



3.1. ábra. Routing algoritmusok csoportosítása [3]

Jelen alkalmazás esetében a hálózati struktúra szerint fogjuk vizsgálni a routing algoritmusokat.

3.1. Hely alapú routing algoritmusok

A hely alapú routing algoritmusok (*location based routing*) helyi információkat használnak fel az üzenetek útjának megtervezéséhez. Ehhez szükséges helymeghatározó eszköz jelenléte az egyes egységeken, mely jelen alkalmazásban a GPS chipek nagy költsége, valamint a beltéri felhasználás miatt előnytelen megoldás lenne.

Egy másik megoldás a jelerősségen alapuló helymeghatározás, mely során a node-ok a fogadott üzenetek RSSI (*received signal strength indicator*) értéke alapján törekednek a helymeghatározásra, azonban ez nem szolgáltat pontos és megbízható adatokat.

Az általunk tervezett rendszer általános felhasználási környezete lehetővé teszi a node-ok helyének előzetes pontos ismeretét, továbbá a kommunikáció szempontjából sokkal fontosabb annak minősége, így eltekintünk a hely alapú routing algoritmusok vizsgálatától.

3.2. Flat routing algoritmusok

A *flat routing* megközelítésben a node-oknak egységes szerep jut a hálózatban, azaz a feldolgozóegység, másnéven ’*base station*’ vagy koncentrátor kivételével nincs kitüntetett szerep, és a node-ok együttműködésén alapul [3]. Az alapkoncepció szerint az ilyen hálózatoknál nem alkalmaznak egyedi azonosítást a nagy számú node-ok miatt, ehelyett adatcentrikus megközelítést alkalmaznak. Ez a jelen okosmérőóra hálózat esetében a korábban leírt felhasználói egyedi azonosítás miatt nem megfelelő megközelítés, azonban az algoritmusok alapelve hasznos alapokat adhat azok módosítására. A következőkben a legelterjedtebb *flat routing* algoritmusokból kerülnek bemutatásra.

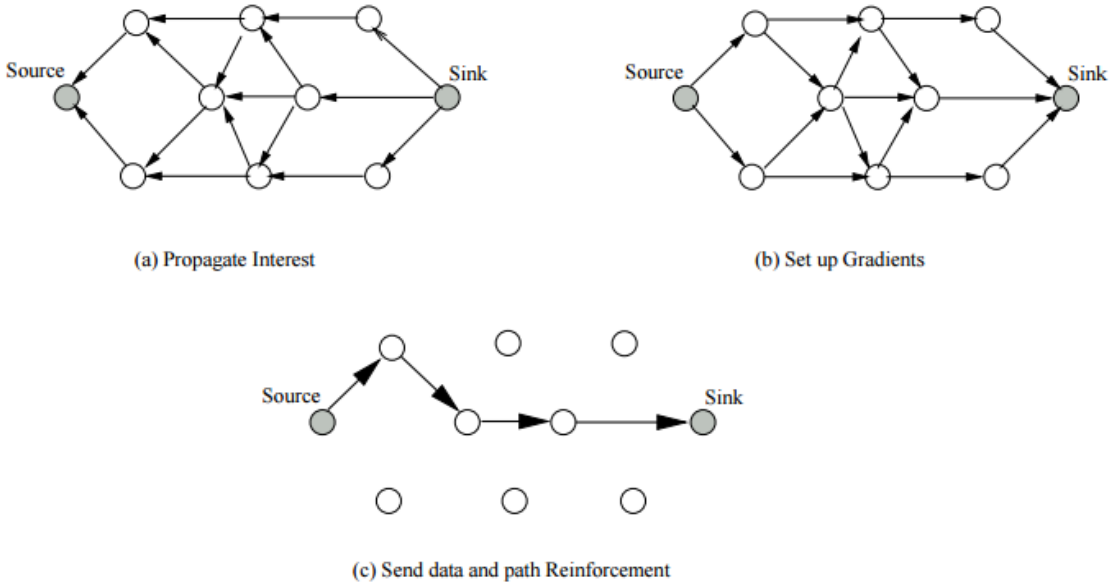
3.2.1. Directed Diffusion

A *directed diffusion* algoritmus esetén a node-ok által generált adatokat kulcs-érték pár szerint azonosítjuk. Az algoritmus alapelve a több forrásból származó adatok kombinálása aggregáció segítségével, az adatredundancia kiküszöbölésével, valamint minél kevesebb kommunikációs overheaddel. Az algoritmus ezen felül képes több forrásból utakat találni egyetlen cél felé [3].

Az algoritmus működése az egyes node-ok a szomszéd node-ok felé felállított gradiensein, valamint a koncentrátor által küldött kéréseken alapszik. A kérések egy feladatot határoznak meg, melyet a hálózatnak teljesítenie kell. Ez a hálózaton a node-ok általi broadcastolással terjed el, mely közben történik a gradiensek felállítása, oly módon, hogy minden, a kérést fogadó node egy gradiens értéket állít fel a küldő felé. Ez a folyamat

addig folytatódik, míg a gradiensek felállnak a koncentrátor felé. Üzenetek küldésekor az üzenetek ezeken a felállított gradienseken keresztül továbbítódnak az áthaladó node-ok üzenetének aggregálásával. A cél egy megfelelő aggregációs fa keresése, mely eljuttatja az adatokat a szenzor node-októl a koncentrátorig. A koncentrátor periodikusan újra küldi a kéréseit az üzenetek fogadásakor, mivel az algoritmus nem biztosítja a kérések megbízható elterjedését a hálózatban [3].

Az alábbi ábra a) részében a kérés terjedését, a b) részében a gradiensek felállítását, a c) részében pedig egy adatküldést láthatunk.



3.2. ábra. Kérés terjedése a hálózaton [3]

3.2.2. Gradient-Based Routing

A *gradient-based routing* a *directed diffusion* eljárás egyik variánsa, mely a gradiens értékeket a koncentrátortól származó kérés hop-száma szerint határozza meg, oly módon, hogy minden node származtat önmagára vonatkoztatva egy magasság értéket, mely a koncentrátortól való hop-távolságát fogja jelenteni. Két node közötti gradiens így a magasságuk különbségéből fog adódni, az üzenetek pedig a legnagyobb gradiens mentén fognak továbbításra kerülni. Az eljárás továbbá alkalmazhat adataggregációt, és forgalomelosztási módszereket is.

Három adatterjesztési mód került definiálásra az algoritmushoz a hálózat energiahatékonyság növelésének érdekében [3]:

- **Sztochasztikus:** abban az esetben, amennyiben egy node több szomszédos nodejának gradiense megegyezik, a továbbítás útja véletlenszerűen kerül kiválasztásra.
- **Energia-központú:** egyes node-ok bizonyos energiaszint alatt növelik a hálózatban betöltött magasságukat, így ritkábban vesznek részt a kommunikációban. Ehhez szükséges az energiaszint megfigyelésének lehetősége.
- **Folyam-alapú:** amennyiben egy node már megtalálható egy adatúton, újabb adatutak nem azon a node-on keresztül fognak áthaladni, így csökkentve a részvételét a hálózatban.

Ezen kiegészítések célja a hálózat kiegyensúlyozott adateloszlása, így növelve a hálózat élettartamát, azonban jelen alkalmazásban nem fontos szempont az energiatakarékosság.

3.3. Hierarchikus routing

A hierarchikus routing eljárások eredetileg vezetékes hálózatoknál alkalmazott eljárások melyek elősegítik a hálózat skálázhatóságát és a kommunikáció hatékonyságát (minek következtében az energiahasználat hatékonyságát is). A hierarchikus architektúrában egyes node-oknak kitüntetett szerep jut, például a nagyobb energiával rendelkező node-ok végezhetnek adatfeldolgozást, vagy a hálózat működéséhez szükséges plusz kommunikációt, míg az alacsonyabb energiával rendelkező node-ok csak méréseket végeznek, illetve minimálisan vesznek részt a kommunikációban [3].

3.3.1. Self Organizing Protocol

A *self-organizing*, azaz önszerveződő protokoll, a szenzor node-okat két csoportba, router node-okra valamint szenzor node-okra bontja. A node-ok elsődleges célja a szomszédjaik felderítése, és routing táblák felépítése mely után egy hierarchikus fa keletkezik [3].

A fa felépülése után a node-ok egyenként címezhetők egy adatkoncentrátor által. Nagy előnye ennek a protokollnak a robosztusság: az új node-ok jelzik a jelenlétüket és bekerülnek egy routing táblába. Ehhez szükséges a node-ok egyedi címezhetősége mely jelen alkalmazásban is követelmény.

3.3.2. LEACH protokoll

A mozaikszó kibontása a *Low Energy Adaptive Clustering Hierarchy*. A protokoll klaszter alapú, mely azt jelenti, hogy a hálózaton belül kisebb csoportosulások, alhálózatok jönnek létre a működés során. A protokoll periodikusan néhány szenzor node-ot véletlenszerűen kiválaszt, melyek ezután a *cluster head*-ek (CH) szerepét töltik be a következő periódusig, és a közelükben levő szenzor node-ok adatait gyűjtik. A protokoll célja az energiafelhasználás, valamint az egyenletes kommunikáció elosztása.

A klaszteren belül TDMA/CDMA közeghozzáférési algoritmusokat alkalmaznak, CH pedig tömöríti a klaszter adatait, illetve továbbítja az adatkoncentrátor felé. Az adatgyűjtés periodikusan történik, így ez az eljárás nem megfelelő olyan hálózatoknál melyek állandó monitorozást igényelnek [3].

A LEACH protokoll működése két fázisból áll: a *setup,* valamint a *steady* fázisból. A *setup* fázisban kerülnek megalkotásra a klaszterek, illetve kerülnek kiválasztásra a *cluster head*-ek. A klaszterek kiválasztása sztochasztikus módon történik. Minden node generál egy véletlenszerű számot, illetve rendelkezik egy előre meghatározott határértékkel, és amennyiben a véletlenszerű szám kisebb, mint a határérték, akkor a node CH-ként üzemel az adott periódusban. Ezután a CH kihirdeti a közeli szenzor node-oknak a létezését, a szenzorok pedig jelerősség alapján eldöntik melyik CH-hoz kívánnak tartozni, melyet egy üzenettel jeleznek a CH-nak. Az összes üzenet megérkezte után a CH elosztja a szenzor node-ok TDMA időszeleteit, és broadcastolja azoknak.

A szenzor node-ok a *steady* fázisban a saját időszeletükben küldhetnek mérési adatot a CH-nak. Egy bizonyos idő után a protokoll újra *setup* fázisba kerül, és új CH kerül kiválasztásra, így elosztva az erőforrásokat.

Ezzel a protokollal több gond is van, miszerint a *cluster head*-eknek pont-pont kapcsolattal kell rendelkezniük a koncentrátor felé, mely sok szenzorhálózati alkalmazásban nem lehetséges. Továbbá a CH-k eloszlásának egyenletességét sem garantálja.

3.4. Konklúzió

Az implementálandó hálózatban több mérőóra node, és egy adatkoncentrátor szerepel. A fő célja a hálózatnak az összes node, adatkoncentrátor által történő elérése. Az egyes node-ok egyéni címezhetősége az alkalmazás szempontjából fontos követelmény, csak úgy, mint a robosztusság.

Ezen szempontok, illetve az előzőleg bemutatott néhány elterjedt módszer alapján a legkedvezőbb tulajdonságokkal a Self Organizing Protocol rendelkezik, a routing táblák alkalmazása kielégíti a címezhetőségi követelményt, továbbá a node-ok egymás közötti kommunikációja lehetőséget ad a hálózat karbantartására is.

A Self Organizing Protocol egy konkrét megvalósítása az alábbi: a hálózat felépülése a node-ok jelenlétének jelzésével indul, mely egy vezérlési információkat tartalmazó üzenet lehet, mely minden node által továbbításra kerül. Az üzenet tartalmazza a küldő egyedi címét, az üzenet hop-számát, továbbá az üzenet továbbításakor hozzácsatolásra kerül a jel erőssége is. Így az egyes node-ok információt szerezhetnek a környezetükről, eltárolhatják a közvetlen elérésű node-ok listáját. Az adatkoncentrátor ezt a listát az egyes node-októl lekérheti, mely segítségével megalkothatja a hálózatot. A hálózat megalkotása az egyes node-ok router node-ként való kijelölését jelenti, mely során elküldésre kerül az adott router node alá tartozó node-ok, valamint a környező router node-ok listája, ezeket routing táblának nevezzük. A node-ok kommunikációjakor acknowledge üzenetekkel jelzik az üzenetek fogadását, ezek elmaradásakor a node-ok azt jelezhetik az adatkoncentrátornak, mely megfelelő lépéseket tehet a hálózat működése fenntartásának érdekében.

Az implementáció során egy ilyen routing protokoll megvalósítása a cél.

4. Biztonság, titkosítás

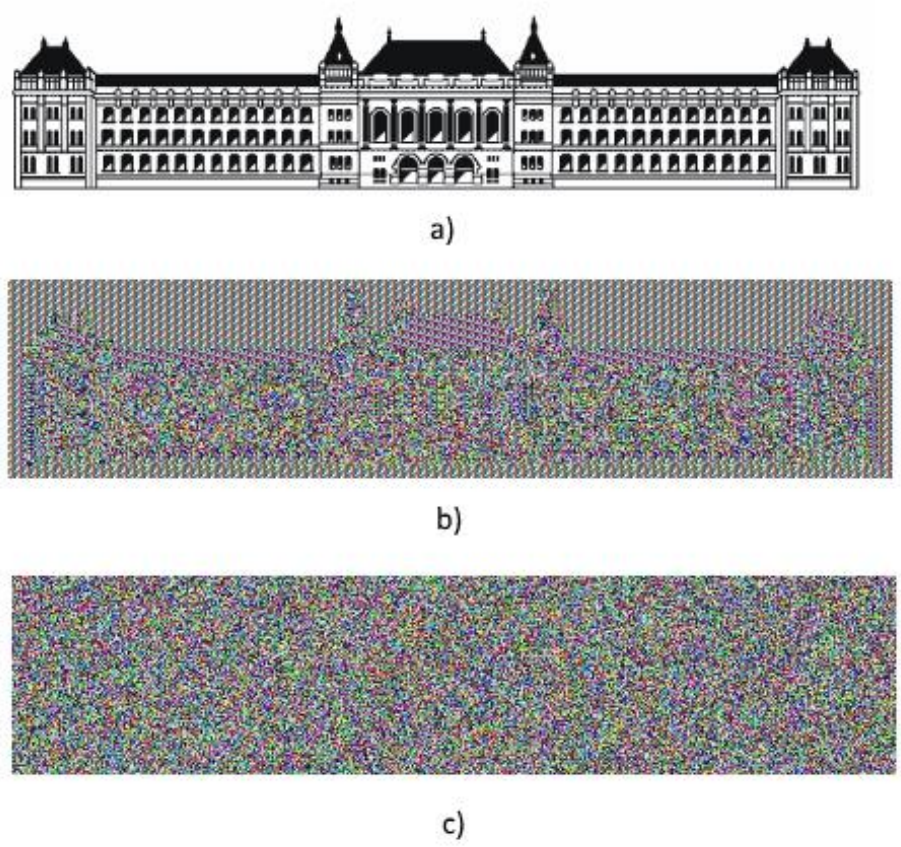
Két adattitkosítási módszert különböztetünk meg: a szimmetrikus, illetve az aszimmetrikus kulcsú titkosítást. A továbbiakban ezen két módszer kerül bemutatásra az alkalmazás szempontjából vizsgálva.

4.1. Szimmetrikus kulcsú titkosítás

A szimmetrikus [4], vagy más néven titkos kulcsú titkosítás ugyanazzal a kulccsal kódolja és dekódolja az üzeneteket. A leggyakrabban használt algoritmusok az AES, DES, Blowfish, RC4; ezek közül megkülönböztetünk folyam- vagy blokk-alapú módszereket.

A folyam-alapú módszereknél a kódolandó digitek egyesével kerülnek kódolásra a kezdeti kulcs és a korábbi kódolt bitek alapján. Az ilyen eljárások előnye a gyorsaság, illetve az alacsonyabb implementációs komplexitás, azonban kevésbé biztonságosak, mint a blokk-alapú eljárások, ha nem megfelelően használjuk őket, például kétszer alkalmazzuk ugyanazt a kulcsot a titkosításra.

A blokk-alapú szimmetrikus kulcsú titkosítások legtöbbször a kulcs bitekben számolt méretnek megegyező blokkokban dolgozzák fel az adatot. AES256 esetén például a 256 bites kulccsal számolt blokkok 256 bit hosszúságúak. Ezeket tovább csoportosíthatjuk iterált, vagy nem iterált módokra. A nem iterált módszernél a kódolandó adat először egyenlő, a kulccsal megegyező blokkokra kerül felosztásra, majd mindegyik blokk egyenként kerül kódolásra. Az így kapott kódolt adatfolyam az eredeti adat jellegét nem rejti el, melyet jól szemléltet a 4.1 ábra. Az eredeti képet (4.1. a) az 1. függelékben szereplő kóddal AES ECB (4.1. b) valamint AES CBC (4.1. c) kódoltuk majd megjelenítettük.



4.1. ábra. a) eredeti kép, b) AES-ECB kódolás, c) AES-CBC kódolás

Az AES ECB kódolás nem iterált módszer, így látható az eredeti kép körvonala, illetve más részletek is. Az AES CBC kódolás már iterált blokkos kódolás, mely kimenetén már nem tapasztalunk az eredeti képhez hasonló formákat.

Ez amiatt van, mivel az iterált, vagy láncolt módszeren alapuló szimmetrikus kulcsú titkosítás a blokkokat egymás után kódolja, úgy, hogy a kódoló kulcsot függővé teszi a korábbi kódolt blokkok kimenetétől. Ezen tulajdonságok és következmények miatt a láncolt szimmetrikus kulcsú titkosításokat ajánlott alkalmazni. Manapság sok eszköz hardveresen támogat AES és DES kódolást, így használatuk egyszerű és gyors, valamint biztonságos.

A biztonsági kockázat a szimmetrikus kulcs elosztásakor jelenik meg, mely jelen alkalmazásban vezetéknélküli módon történik, így annak lehallgatása könnyen megtörténhet. Ennek kiküszöbölésére szükséges kulcselosztási algoritmusokat használni, melyekből kettő a 4.3. fejezetben kerül ajánlásra az alkalmazáshoz.

4.2. Aszimmetrikus kulcsú titkosítás

A nyilvános kulcsú titkosítás [5] előnye az aszimmetrikus kulcsú titkosítással szemben, hogy a kódoló (küldő) és a dekódoló (fogadó) félnek semmilyen titkos jelszót vagy kulcsot nem kell cserélnie egymással. Ehelyett minden egyes felhasználó rendelkezik egy kulcspárral, mellyel a biztonságos kommunikáció létrejöhet. Az egyik kulcsot privát kulcsnak, a másikat publikus kulcsnak nevezzük. Ezen kulcspár generálása ugyanazon eljárás során jön létre, azonban ezek egymásból nem következtethetők ki. A privát kulcs titokban tartása létfontosságú a biztonság szempontjából, ezzel ellentétben a publikus kulcsot a kommunikáció minden résztvevőjével ismertetni kell. Így az aszimmetrikus titkosítás esetén a kommunikációban résztvevőknek nincs szükségük megbízható csatornára, csak a publikus kulcs hitelességét, a megfelelő tulajdonosához való tartozását kell bizonyítaniuk.

A titkosítás során a kódolt adatot fogadó oldalhoz tartozó nyilvános kulccsal történik a kódolás, míg a dekódolás a vevő oldalon csak a titkosításhoz használt publikus kulcshoz tartozó privát kulccsal lehetséges.

A hátránya ezeknek a titkosítási módszereknek a szimmetrikus kulcsú titkosítással szemben a számításigényük, azonban a kulcselosztási probléma nem áll fenn.

4.3. Titkosítási megoldások a mérőóra-hálózatban

Összefoglalva az előző két titkosítási módszert bemutató fejezetet elmondható, hogy bár a publikus kulcsú titkosítás a kulcspárok használata miatt kiküszöböli a kulcselosztási problémát, azonban a nagyobb számításigény és a hardveres támogatás hiánya nem teszi megfelelővé használatát az alkalmazásunkban. A beágyazott eszközökben található szimmetrikus kulcsú titkosítást támogató hardverek költsége alacsony, illetve kevés számításigénnyel rendelkeznek, azonban jobban ügyelnünk kell a biztonsági kérdésekre. Megoldás lehet egyetlen, az elemek gyártásakor beégetett kulcs használata, azonban egy kulcs használata nem ajánlott a *brute-force* támadások miatt. Ezen problémák összevetése után két egyszerű megoldás tárulkozik elénk.

4.3.1. Kevert megoldás

Az első megoldás kihasználja mindkét módszert: az adatokat szimmetrikus kulcsú eljárással titkosítjuk mely a hálózat gyorsaságát segíti elő, a kulcselosztást azonban a lassabb, számításigényesebb publikus kulcsú titkosítással végezzük, illetve bizonyos periodicitással új kulcsot generálunk és osztunk szét a hálózatban ily módon. Egy hasonló módon történő, kulcsszervert alkalmazó elosztott autentikációs eljárás folyamata a következő ('P' a node, 'Q' az adatkoncentrátor, 'CA' a tanúsítványkezelő, illetve 'k' egy adott szereplő publikus kulcsa, 'k^{-1}' a privát kulcsa) [6]:

- **P → Q**: 'P' autentikációt kezdeményez 'Q' felé
- **Q**: nonce[1] generálása
- **Q → P**: a nonce elküldése (n)
- **P**: üzenet számítása: $m = [P, Q, n]k_p^{-1}$, aláírja a privát kulcsával
- **P → Q**: m elküldése
- **Q → CA**: 'P' publikus kulcsának lekérése a szervertől
- **CA**: az adatbázisból kp lekérése, majd a 'c' tanúsítvány generálása ($c = [P,k_p]k_{CA}^{-1}$)
- **CA → Q**: P és c küldése
- **Q**: P, k_p visszafejtése 'c'-ből k_{CA} segítségével, majd $[P, Q, n]k_p = [m]k_p$ ellenőrzése. Amennyiben ez az 'm' és a P által küldött 'm' megegyezik, az autentikáció sikeresnek minősített.

[1] A nonce a kriptográfiában egy egyszer használatos (általában pszeudorandom) szám, mely az ismétléses támadások ellen véd

4.3.2. Egyszerűsített megoldás

Egy elterjedt, csak szimmetrikus kulcsú titkosításon alapuló autentikációs eljárás a Needham-Schroeder protokoll [7], mely egy kulcsszervert is alkalmaz. Az általunk fölvázolt – hasonló alapgondolatokra épülő – egyszerűsített kulcselosztási algoritmus esetén az egyes egységek egy gyártáskor beégetett titkos azonosítóval rendelkeznek, melyet egy adatkoncentrátor is ismer. Ezután a kulcselosztás úgy történik, hogy az adatkoncentrátor broadcastol egy *nonce*-ot (random bitsorozatot), melyből a node-ok a titkos azonosítójukat felhasználva matematikai transzformációval egy egyedi titkos kulcsot generálnak. Ezt az adatkoncentrátor is elvégzi minden egyes ismert titkos azonosítóra, így az adatok kódolása és dekódolása egységenként megtörténhet szimmetrikus kulcsú módszerrel. Ez a megoldás kiküszöböli a kulcselosztásból származó biztonsági nehézséget, azonban a hálózatban szereplő mérőórák egyedi kulcsának adatkoncetrátor általi ismeretét igényli.

A publikus kulcsú titkosítás számításigénye, illetve a hardveres támogatás hiánya miatt jelen rendszer implementációjában ezen megoldás implementálására fogunk törekedni.

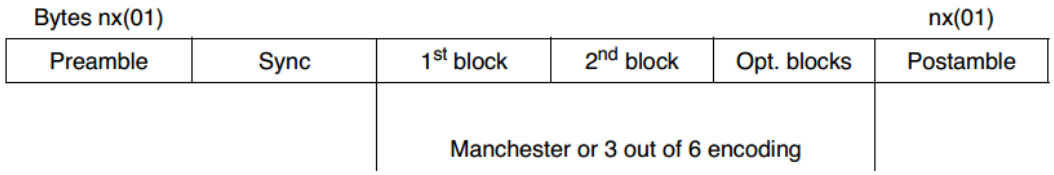
5. A vezetéknélküli MBUS szabvány

Az EN 13757-4-es számú szabvány a vezetéknélküli alkalmazást és több működési üzemmódot határoz meg, melyek többek között definiálják az adatsebességet, további fizikai paramétereket, az adatok kódolását, és az egyes adatcsomagok mezőinek jelentését. Mivel az M-Bus nem hálózat, ezért az adatokat csak szolgáltatja és fogadja, a nagy hálózatokon történő szállítást a protokoll nem tartalmazza (a hét OSI rétegből a *Transport* a *Session* és a *Presentation* üres) [1].

Az alábbi fejezetben bemutatásra kerül a wM-Bus vezetéknélküli okosmérőóra szabvány adatkapcsolati rétege, mely az implementáció során felhasználásra kerül.

5.1. A wM-Bus csomag felépítése

Egy szabványos csomag a következő mezőkből épül fel:



5.1. ábra. Egy M-Bus csomag felépítése [1]

A *'preamble'* mező '01' bit sorozatok egymásutánja, melyek száma a szabvány szerint megadott minimum és maximum mennyiség között lehet. Ez függ a választott WM-Bus üzemmódtól. Ezt követi egy szinkronizációs mező, melynek értéke csakugyan a később bemutatott üzemmódoktól függ, szerepe a *preamble* végének jelzése.

Ez után következnek a tényleges adatmezők. Az első és második adatblokk mindegyik üzemmódban szerepel, illetve a szabvány engedélyez további opcionális adatblokkokat is. Ezeknek a kódolása vagy Manchester-kódolással, vagy *„3 out of 6"* kódolással történik, ami 4 bites csoportokat 6 bitesen kódol egyenlő számú nullákkal és egyesekkel, minimum két 0-1 vagy 1-0 ugrással (példaképp a 000111 tehát nem érvényes kódszó).

NRZ-Code	Decimal	6 bit code	Decimal	N° of transitions
0000	0	010110	22	4
0001	1	001101	13	3
0010	2	001110	14	2
0011	3	001011	11	3
0100	4	011100	28	2
0101	5	011001	25	3
0110	6	011010	26	4
0111	7	010011	19	3
1000	8	101100	44	3
1001	9	100101	37	4
1010	10	100110	38	3
1011	11	100011	35	2
1100	12	110100	52	3
1101	13	110001	49	2
1110	14	110010	50	3
1111	15	101001	41	4

5.2. ábra. A 3 out of 6 kódolás kódtáblája [1]

Végül egy *'postamble'* mező következik, mely a *'preamble'* mezőhöz hasonlóan egy '01' sorozat, a szabvány és az üzemmódok alapján.

Az első üzenetblokk felépítése a következő:

L-field	C-field	M-field	A-field	CRC-field
1 byte	1 byte	2 bytes	6 bytes	2 bytes

5.3. ábra. Az első blokk felépítése [1]

A keret első byte-ja a hossz mező (0-255), mely jelzi az összes adatbyte számát, leszámítva a hossz és a CRC mezőket. Amennyiben (L-9) MOD 16 nem 0, az utolsó blokk (L-9) MOD 16 adatbyte-ot és 2 CRC byte-ot fog tartalmazni.

A második byte a kontrol mező, mely a keret típusát jelzi, az ezután következő M mező pedig egy egyedi, gyártó-specifikus azonosítót tartalmazó bytekettős. Ebben az esetben először az alacsonyabb byte-ot küldjük el. Amennyiben a felső két MSB nulla, a 6 byte-os 'A' (*address*) mező egy gyártó által biztosított globálisan egyedi azonosító lesz. Ha a két MSB nem nulla, az *address* mezőnek legalább a küldési hatótávon belül egyedinek kell lennie.

A második üzenetblokk a következőképp épül fel:

CI	Data	CRC
1 byte	15 byte or ((L-9) modulo 16) –1 if it is the last block	2 byte

5.4. ábra. A második blokk felépítése [1]

Az első byte a CI azaz *Control Information* mező, mely jelzi az utána következő adat típusát a felsőbb rétegek felé.

Ezt követhetik további opcionális adatblokkok is.

Data	CRC
16 byte or ((L-9) modulo 16) if it is the last block	2 byte

5.5. ábra. Opcionális adatblokk felépítése [1]

Minden keret végén 2 byte-nyi CRC érték szerepel, mely az IEC 60870-5-1 szabvány $x^{16} + x^{13} + x^{12} + x^{11} + x^8 + x^6 + x^5 + x^2$ polinomjával számítandó.

5.2. A wM-Bus üzemmódok

A definiált **S1 mód** esetén a node csak küldőként funkcionál, egy koncentrátor felé. Stacionárius adatkoncentrátor esetén célszerű lehet a vevő energiafelhasználását is figyelembe venni, így ekkor hosszú fejléc (*preamble*) használata ajánlott, időt adva így az üzenetet vevő node felébredéséhez, mivel a vevő akkumulátoros táplálású is lehet, és megtörténhet, hogy épp alacsony energia-felvételű állapotban van. Az **S1-m** almód a mobilleolvasók felé történő üzenetküldés esetén definiált, folyamatosan bekapcsolt vevővel, így a fejléc lehet rövid. Az **S2** mód folyamatosan bekapcsolt stacionárius vevőt, vagy szinkronizált hálózatot feltételez, így ebben az esetben is elég rövid fejlécet alkalmazni.

Az adatmezők kódolása Manchester-kódolással történik, azaz minden bitet két bittel írunk le, mikor is az 1→0 átmenet jelenti a ’0’ bitet, illetve a 0→1 átmenet az ’1’ bitet. Minden adatbájt MSB-je kerül először továbbításra, míg több byte-os adatmezőknél az alsó byte, CRC esetén a felső byte [1].

A *preamble* a következőképp néz ki: S1 mód esetén ***n*** × *('01')*, ahol n >= 279 (*long header*), **S2** mód esetén pedig n >= 15 (*short header*). Az ezt követő szinkronizációs blokk értéke 0001110110 10010110.

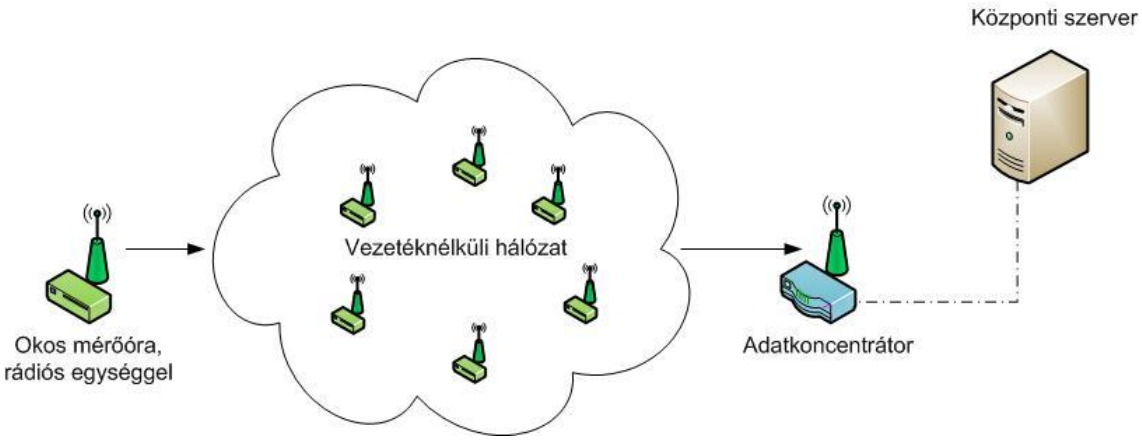
T1 mód esetén a küldés rövid sorozatszerű adatokkal történik pár másodpercenként, **T2** esetén pedig küldés után rövid időre vevő állapotba lép a node és fenntartja a kapcsolatot a válaszadóval. Az adatok kódolása *„3 out of 6"* módon történik a teljesítmény javítása érdekében, azonban a Manchester-kódolás is megengedett annak egyszerűsége miatt. Az adatátvitel MSB first módon történik, míg CRC esetén az alsó byte, gyártó ID esetén pedig a felső byte küldése történik először.

A *'preamble'* illetve az azt követő szinkronizációs mező értéke n × ('01') + 0000111101, ahol n>=19. A *'preamble'* szerepe a mintavételezés szinkronizációjának elősegítése a vevő számára.

Megemlítendő még az **R2** üzemmód, mely gyakori bidirekcionális kommunikációt tesz lehetővé *meter* és *other* eszközök között.

6. Rendszertervezés és implementáció

A megvalósítandó rendszer sematikus felépítése a 6.1. ábrán látható.



6.1. ábra. A rendszer sematikus felépítése

A rendszer elemei a következők:

- Mérőóra node-ok, és az azokból felépülő hálózat
- Adatkoncentrátor, mely a központi szerver és a vezetéknélküli mérőórahálózat között hoz létre kapcsolatot (gateway)
- Központi szerver, mely az adatok feldolgozását végzi

A mérőóra node-ok és az adatkoncentrátor is rendelkezik rádiós egységgel mely segítségével létrejön a vezetéknélküli hálózat. Az adatkoncentrátornak továbbá rendelkeznie kell további kommunikációs interfésszel a központi adatfeldolgozó szerverhez való kapcsolatra, mely esetünkben GSM kommunikáció.

Az alábbi alfejezetekben a rendszer elemeinek tervezése és implementálása kerül részletezésre.

6.1. A rádiós egység

Az ezen dolgozatban felvezetett rendszer fő alapeleme az elosztott rádiós kommunikáció, mely magával hordozza az hálózat felépítését, karbantartását, és biztonságát melyek követelményei a 2. fejezetben szerepelnek. Az alábbi fejezetben az ezt lehetővé tevő rádiós egység követelményei kerülnek összefoglalásra, valamint bemutatásra kerülnek piacon már elérhető megoldások melyek ezeket részben, vagy egészben teljesítik, továbbá a választott egység működése is ismertetésre kerül.

6.1.1. Követelmények

A rendszerben alkalmazott rádiós egységnek a 868 MHz-es ISM sávban kell működnie, mely az európai szabadfelhasználású frekvenciasáv.

A feladat követelménye a wM-Bus szabvánnyal való kompatibilitás, mely a rádiós kommunikáció fizikai és adatkapcsolati rétegét határozza meg. A szabvány hálózati réteget nem definiál, így annak implementálása szabadon választható, mindössze a változó csomagméret támogatása követelmény.

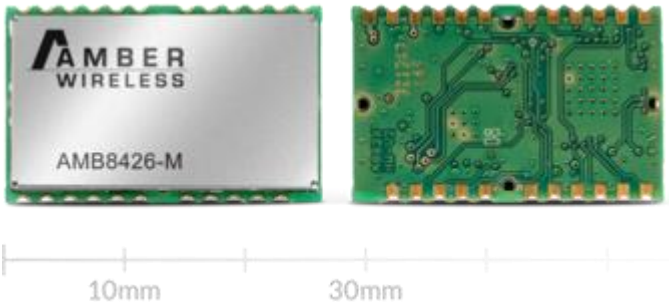
6.1.2. Piackutatás

A legtöbb nagy félvezetőgyártó palettáján megtalálhatók rádiós chip-ek, melyek funkcionalitása között a Wireless M-Bus is szerepel. Ilyen például a Texas Instruments CC112x valamint CC120x családja [8], valamint az STMicroelectronics Spirit1 [9] egysége. Az ilyen rádiós egységek legtöbbször a kommunikációnak csak a fizikai, valamint adatkapcsolati rétegét valósítják meg, így alkalmasak egyedi fejlesztésekre.

Már elérhetők SoC (*system on a chip*) kiszerelésű rádiós egységek is, melyek egy chipen tartalmaznak egy soros interfésszel rendelkező vezérlőegységet, valamint a kommunikációhoz szükséges rádiós chipet. Az ilyen egységekhez a vezérlő egységen futó, a gyártó által támogatott *firmware* tartozik, mely magasabb szintű funkcionalitást tesz lehetővé, mint például az adatkapcsolati réteg interfészelése egyszerűbb adatcsomagokkal, vagy további OSI rétegek implementálása.

Mivel széleskörűen elérhetők ezen SoC megoldások, valamint a fejlesztési időt is jelentősen le tudják csökkenteni, ezért a továbbiakban ezek közül kerül bemutatásra néhány gyártó egysége.

6.1.2.1. Amber Wireless rádiós egység

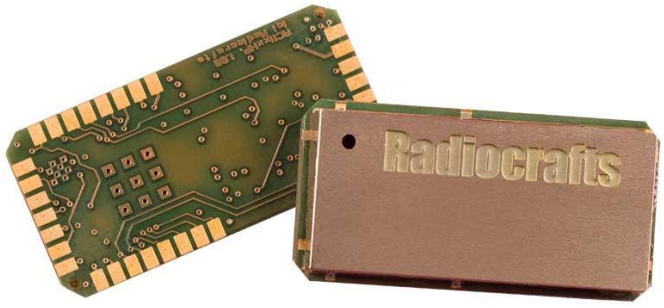


6.2. ábra. Az AMB8426-M modul

Az Amber Wireless egy bluetooth illetve rádiós SoC modulokat gyártó cég, melynek kínálata között találunk kifejezetten vezetéknélküli M-Bus modult is. Az AMB8426-M [10] egy 868 MHz-en működő, az EN13757-4:2013 Wireless M-Bus szabványt követő egység, mely szabad téren 700m-es hatótávval rendelkezik. Konfigurálása és használata UART/SPI interfésszel, valamint további dedikált lábakkal lehetséges.

A konfigurációs regiszterek segítségével beállíthatjuk az UART paramétereket, a kívánt wM-Bus üzemmódot (R, T, S, C), a low-power paramétereket (*auto sleep*), sugárzási teljesítményt, valamint további M-Bus specifikus paramétereket. A modul támogat AES-128 titkosítást is, az eszközben 64 cím-kulcs páros tárolható el. Hálózati réteg támogatás nem található a modul funkciói között, így ezen egység választásakor annak implementálása megoldandó feladat.

6.1.2.2. Radiocrafts rádiós egység



6.3. ábra. A Radiocrafts RC-1180 modul

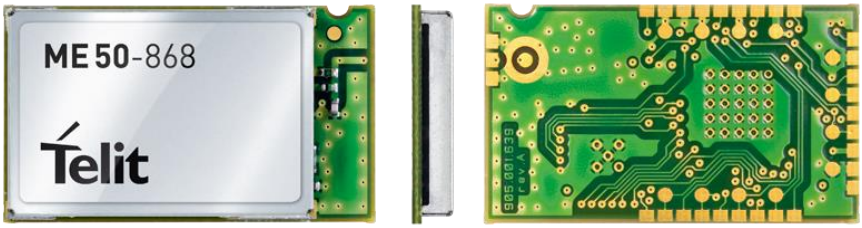
A Radiocrafts egy norvég *smart-metering* és IoT specifikus SoC modulokat gyártó cég, portfóliójában találhatóak ZigBee, KNX, TinyMesh, valamint wM-Bus [11] egységek is.

Az előző fejezetben ismertetett Amber modulhoz hasonlóan a Radiocrafts RC-1180 modulja is megfelel az EN13757-4 szabványnak, különböző beállítások érhetők el itt is, csakugyan UART interfészről vezérelve, illetve csakugyan támogat AES titkosítást.

A modul sajátossága, hogy képes mikrovezérlő nélküli működésre is. Egyes mérőórák a mérési adataikat pulzusokba kódolva szolgáltatják, melyeket a modul a pulzus bemenetein képes feldolgozni és továbbítani. A wM-Bus applikációs rétegben specifikált bizonyos rádiós üzenetek hatására az egység önállóan képes beavatkozni a szelepvezérlő kimenetén is.

Bár a gyártó rendelkezik a TinyMesh technológiával, mely képes robosztus és öngyógyító multi-hop hálózat felépítésére, hasonló funkcionalitásra a wM-Bus egység nem képes, így ezen követelmény implementálása egy külső mikrovezérlő feladata lenne.

6.1.2.3. Telit rádiós egység



6.4. ábra. A Telit ME50-868 modul

A Telit egy rádiós és GSM modulokat gyártó olasz cég, portfóliójában megtalálhatók rövid hatótávú ZigBee, SIGFOX, Bluetooth, valamint Wireless M-Bus [12] egységek is.

Az ME50-868 az EN13757-4, illetve -5 szabványokat elégíti ki. Segítségével egyszerűen megvalósítható egy-, illetve kétirányú kommunikáció gáz-, víz-, hőmérőórákhoz, valamint adatkoncentrátorokhoz, továbbá AES titkosítást is támogat. A modul *a low-power* kategóriába sorolható, 25 mW sugárzási teljesítmény mellett fogadáskor 26 mA-t, küldéskor 45 mA-t fogyaszt, továbbá lehetőség van külső, és periodikus felébresztésre is. A vezérlése TTL RS232 soros interfészen történik AT parancsokkal[2].

[2] Az AT parancskészlet modemek karakteres vezérlésére kialakított szabvány, az 1980-as években került kifejlesztésre, a mai napig támogatják különböző eszközök.

A már adottnak tekintett adatkapcsolati és fizikai rétegen felül az EN13757-5 szabványban leírt, az alacsony hatótávokat kiküszöbölő R2 mód segítségével megvalósított *relay* funkciót is támogatja, mellyel megvalósítható multi-hop hálózat is.

6.1.2.4. Konklúzió

Mindhárom egység magas szintű kezelést tesz lehetővé, melynek köszönhetően a wM-Bus módok kezelése, illetve a csomagok összeállítása leegyszerűsödik, azonban a Telit modul által támogatott multi-hop hálózati lehetőség – mely a 3. fejezetben ismertetésre került Self Organizing Protocol-t valósítja meg a 3.4. fejezetben leírt routing táblák segítségével – kiemeli azt a többi hasonló SoC modul közül, így ez került kiválasztásra a rendszerben való alkalmazáshoz.

6.1.3. A választott egység működése

A gyártó által támogatott firmware az EN13757-5 Wireless M-Bus szabvány szerinti multi-hop kommunikációt valósítja meg, az EN13757-4 szabványban leírt R2 mód által definiált relay-ek felhasználásával.

A relay-ek egyszerre el tudják látni a szabványban leírt *meter* és *other* funkciókat, azaz ha egy üzenet címzettje nem a fogadó eszköz, akkor *other* funkcióból *meter* funkcióba vált, és továbbküldi az üzenetet. A hálózat működése így hierarchikus, és egy fa struktúrával jellemezhető, melyben egy adatkoncentrátor a fa gyökere, az egyes *meter* node-ok a fa levelei, melyek vagy közvetlenül, vagy egymáson keresztül kommunikálnak az adatkoncentrátorral. A *downstream* kommunikáció esetén a relay eszközöket a *meter* node-ok *other* eszközként látják, *upstream* kommunikáció esetén pedig *meter* eszközként [12].

6.5. ábra. A kommunikáció irányainak elnevezése

A hálózat topológiájának ismeretére csak az adatkoncentrátornak van szüksége, melyhez az összes alá tartozó node címét ismernie kell, a relay-eknek pedig csak az alájuk tartozó node-ok címét kell ismerniük. Ezt a nyilvántartást routing táblák alkalmazásával éri el a hálózat. Az adatkapcsolati rétegben a Wireless M-Bus üzenetkeret csak egy címet definiál, ezért a multi-hop kommunikációhoz annak kiegészítésére van szükség. Az adatmező első része tartalmazza az ehhez szükséges információkat, melynek jelenlétét a CI mező 0x81 értéke jelez [12].

Annak érdekében, hogy a koncentrátor bármely másik node-ot elérje, a 13757-5 szabvány egy hálózatmenedzsment protokollt ír elő azon node-ok felderítésére, melyeket a koncentrátor közvetlenül nem ér el. Ezt a folyamatot a modulok önmaguk képesek követni és véghezvinni, így külső beavatkozásra nincs szükség, a felhasználó úgy látja az üzeneteket, mintha a koncentrátor pont-pont kapcsolatban kommunikálna minden mérőóra egységgel. Ehhez a koncentrátornak szükséges minden node címét ismernie, melyet a szabvány és a firmware által nyújtott *installation mode* (installációs) üzenet segítségével ér el.

Annak érdekében, hogy a koncentrátor képes legyen a hálózatban szereplő összes node-al kommunikálni, az üzenetek irányának függvényében két mechanizmussal alakítja a hálózat topológiáját. *Downstream* (koncentrátor → meter) irány esetén a koncentrátor az üzenetbe csomagolja annak teljes útinformációját, *upstream* (meter → koncentrátor) küldési irány esetén pedig a relay-ekhez rendelt routing táblák határozzák meg az üzenet útját. Minden relay-hez két tábla tartozik: az egyik tábla a relay node alá tartozó node-ok címeit, a másik pedig a többi relay node címét tartalmazza. Az első relay a cél-node és a

koncentrátor között hálózati információt ad hozzá a csomaghoz a CI mező 0x81 értékkével. Ezzel megkülönböztethetőek a továbbított és a direkt üzenetek. Egy relay csak akkor továbbítja (ezt nevezzük hop-nak) a hálózaton hozzá beérkező üzenetet, ha a cél-node megtalálható a listájában. Ez alól két kivétel van: egy relay mindenképp továbbítja az üzenetet, ha a táblája üres, valamint ha az üzenet installációs típusú. Upstream esetben az üzenet mérete korlátozza a hop számot, mivel minden hop esetén hozzácsatolásra kerül a hálózati információ mely 11 byte hosszúságú, így ha az üzenet hossza eléri a maximálisan megengedett méretet, azaz 255 byte-ot, a csomag eldobásra kerül [12].

Az üzeneteket fogadó egység mindkét irányban minden hop-nál *acknowledge* üzenetet küld. Ha ennek küldése/fogadása elmarad a küldő automatikusan újraküldi az üzenetet maximum kétszer.

A koncentrátor az ismert node-ok listáját automatikusan kiterjeszti új installációs üzenet fogadásakor. Ha a cél node nem érhető el, vagy ha az üzenethez hozzáfűzött hálózati információk miatt annak mérete meghaladja a megengedettet, a küldeni kívánt üzenet eldobásra kerül [12].

A modul használatának egyik óriási előnye, hogy ezen folyamatokat szinte teljesen elrejti a felhasználó elől, mindössze az installációs üzenet elküldésével kell foglalkoznunk, mely megérkezte után pont-pont kapcsolatként tekinthetünk a kommunikációra.

Az alábbi alfejezetben a kívánt működés eléréséhez szükséges regiszter beállítások, és a hálózati protokoll működése kerül ismertetésre.

6.1.3.1. Konfiguráció

Az eszköz konfigurálása és működtetése UART interfészen keresztül történik ’AT’ parancsokkal melyek az AT karakterekkel kezdődnek, és kocsi vissza (<CR>) karakterrel végződnek. A parancs (*Command*) mező lehet paraméter típusú, mely értékek beállítására és kiolvasására szolgál, valamint ’akció’ típusú. Az egyetlen kivétel a ’+++’ karaktersorozat, mely az AT parancs módba történő belépést jelzi.

A	T	Command	Additional data	<CR>

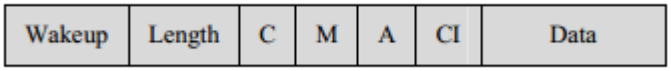
6.6. ábra. Az AT parancskeret [12]

Az ’ATS400=4<CR>’ karaktersorozat elküldésével például a modul regisztertáblájában található 400-as regisztert 4 értékbe tudjuk állítani, melyre a modul sikeres végrehajtás

esetén *OK*, hiba esetén pedig *ERROR*-al válaszol. Bitfieldek esetén azok decimális reprezentációja értendő az érték alatt. Ugyanezen regiszter értékének lekérdezése pedig az ’ATS400?<CR>’ paranccsal történik, melyre a modul ebben az esetben ’S400=4’-el válaszolna. További hasznos AT parancs az *ATR* mellyel a regiszterek értékét alapállapotba lehet helyezni, az *ATT* mellyel folyamatos küldést szimulálhatunk, valamint az *ATO*, mely működési állapotba helyezi a modult, kilépve az AT módból [12].

A kívánt működés eléréséhez szükséges konfigurációs lépések a következők: 400-as regisztert 8 értékbe állítva R2-meter, 9 értékbe állítva pedig R2-other (koncentrátor) wM-Bus módba konfigurálhatjuk a modult, a 454-es regiszter 1, illetve 2 értékével pedig a modul által támogatott hálózati protokollt lehet engedélyezni. A koncentrátorként (*other*) beállított eszköz lesz a hálózati fa struktúra gyökere, míg a *meter*-ként beállított eszközök a node-ok. Két regiszter (411, 418) segítségével megadhatjuk az egység wM-Bus szabványban leírt M-, és A-field-jeit, melyek együtt a node-ok egyedi azonosítóját (címét) adják.

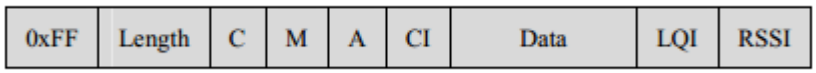
Az adat üzemmódba történő lépés (*ATO*) előtt szükséges a soros kommunikációs adatkeret beállítása is, mely az üzenet formátumát határozza meg. A felhasználó → Telit irányú üzenetek beállítását a 401-es regiszteren keresztül érhetjük el, itt egyes mezők jelenléte konfigurálható. Az általunk beállított formátum minden lehetséges mezőt alkalmaz:



6.7. ábra. A küldési adatkeret [12]

- *Wakeup*: 0xFF vagy 0x00 karakter mely a frame kezdetét jelzi
- *Length*: az ez után következő karakterek száma
- C: a frame szerepét jelzi (Request, ACK, stb.)
- M és A: koncentrátor esetén a címzett M-fieldje (2 byte) és A-fieldje (6 byte), meter esetén a saját M, és A-fieldek
- CI: vezérlési információt meghatározó karakter
- *Data*: payload

A rádiós egységtől az azt vezérlő kontroller irányába haladó soros adat üzenetek formátuma a 402-es regiszteren keresztül állítható:



6.8. ábra. A fogadási adatkeret [12]

- *Wakeup*: 0xFF karakter mely a frame kezdetét jelzi
- *Length*: az ez után következő karakterek száma
- C: a frame szerepét jelzi (Request, ACK, stb.)
- M és A: koncentrátor esetén a küldő címe, meter esetén a saját címe
- CI: vezérlési információt meghatározó karakter
- *Data*: payload
- LQI: a rádiókapcsolat minőségét jelző szám (0 - rossz, 3 - kiváló), általunk a kommunikációhoz nem használt
- RSSI: a rádiójel előjeles dBm-ben kifejezett erőssége, általunk a kommunikációhoz nem használt

Mint már említésre került, a modul támogatja az AES-128 titkosítást, azonban a kulcsok regisztertáblában történő tárolása csak konfigurációs üzemmódban érhető el. Mivel a számunkra szükséges kulcselosztás a rádiós kapcsolaton, normál működési állapotban történik (melyből kilépés után újra fel kell építeni a hálózatot), valamint mivel a modul által támogatott eltárolható cím-kulcspárok száma 32-re korlátozott, a modul ezen funkciója nem kerül felhasználásra.

6.2. A hálózati applikációs protokoll

A rádiós egység kiválasztása és bemutatása után definiálásra kerülhet a hálózat részletes működése.

Mint az már említésre került, a Telit ME50-868 a wM-Bus EN13757-5 szabványban leírt R2 módban megvalósított *Wireless relaying* segítségével implementálja a hálózati protokollját. Ez azt jelenti, hogy az egyes node-ok képesek *meter*-ként, valamint *other*-ként is üzemelni, mely lehetővé teszi az üzenetek továbbítását. Mind a mérőóra node-ok, mind az adatkoncentrátor ezt a rádiós modult használják.

A 6.1.3. fejezetben kifejtésre került a modul által elrejtett hálózati működés, azonban a rendszer specifikálásához még szükséges a mérőórák és a koncentrátor közötti kommunikációs üzenetek definiálása.

Az alábbiakban a mérőóra node-ok installációja, a kulcselosztás, valamint a mérési adat begyűjtése kerül bemutatásra. A saját üzenetek definiálásakor a wM-Bus szabvány szerinti üzenetkeret CI (*control information*) mezőjét használjuk az üzenetek applikációs jelentésének jelzésére.

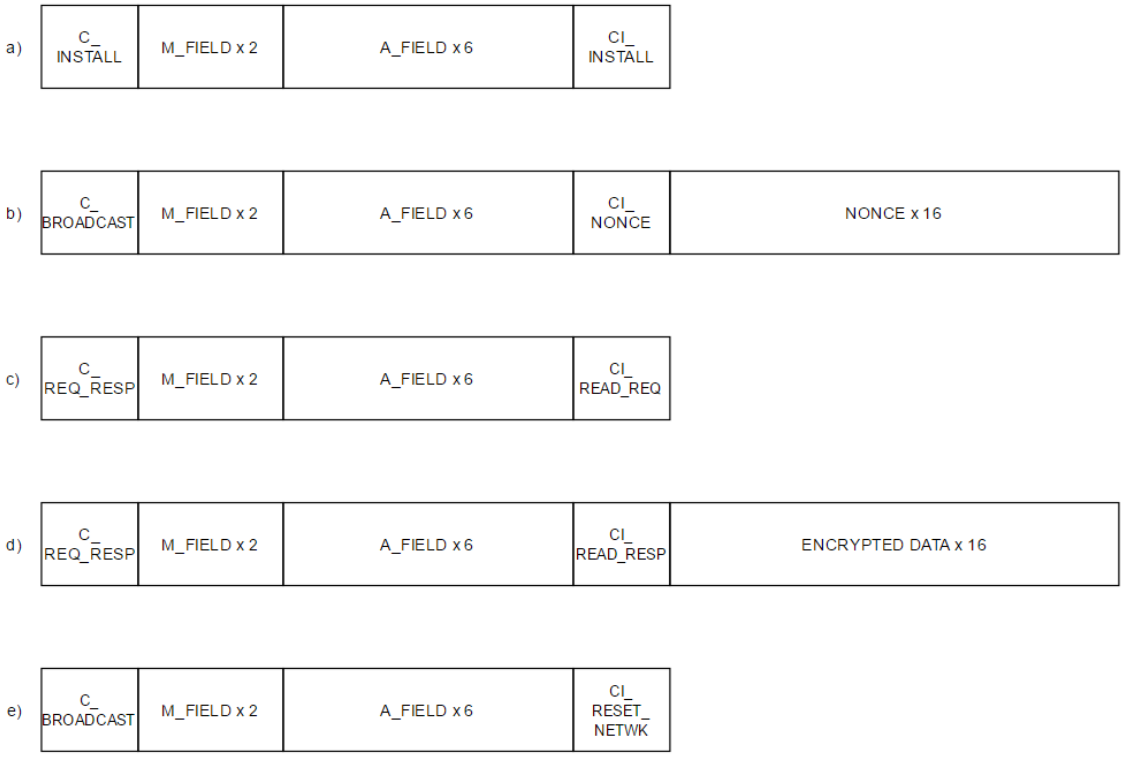
A hálózatban adottnak tekintjük az adatkoncentrátor mindenkori jelenlétét, ellentétben a mérőóra node-okkal. A node-ok jelenlétüket a 6.1.3. fejezetben ismertetett installációs üzenetekkel jelzik az adatkoncentrátor felé, a felhasználónak (jelen esetben a modult vezérlő mikrokontrollernek) azonban ezt manuálisan kell elküldenie. Fogadásakor az adatkoncentrátor automatikusan regisztrálja ezeket a node-okat, és bár nem küld automatikus visszajelzést, az üzenetet továbbítja a felhasználónak.

Jelen rendszerben egy node a bekapcsolás után percenként elküldi az installációs üzenetét (CI_INSTALL = 0x06) mindaddig, amíg az adatkoncentrátortól nem kap neki címzett üzenetet (azaz bekerült a hálózatba). Ezt az időtartamot a node-ok installációs fázisának nevezzük. Az adatkoncentrátor a hálózat hibatűrésének javítása érdekében periodikusan újraépíti a hálózatot, mely után adott ideig vár az installációs üzenetek beérkezésére (ezt nevezzük az adatkoncentrátor installációs fázisának). Az újraépítést (*reset*) a node-oknak egy CI_RESET_NETWK (0x80) üzenettel előre jelzi, melynek hatására azok újra installációs fázisba lépnek. Az adatkoncentrátor vezérlőben csak az installációs fázisa alatt beérkezett installációs üzeneteket vesszük figyelembe oly módon, hogy ezen node-ok adatai kerülnek majd lekérdezésre (mely hatására nem küldenek több installációs üzenetet). Azon node-oknak, melyek ez után szeretnének belépni a hálózatba, a következő újraépítésig és installációs fázisig várniuk kell. Az újraépítés naponta egyszer történik meg.

Az adatkoncentrátor az installációs fázis végeztével broadcastolja a napi *nonce* bytesorozatot a 4.3.2. fejezetben leírt szimmetrikus kulcselosztási eljárásnak megfelelően. Ez után adatgyűjtési módba kerül, mely során az installációs fázis alatt beregisztrált node-ok mérési adatait egyesével begyűjti egy CI_READ_REQ = 0x78 üzenettel. Az M-, és A-fieldekbe az egyes mérőórák M-, és A-fieldjei kerülnek, melyek segítségével a hálózati protokoll el tudja juttatni az üzenetet a megfelelő node-hoz. Az egyes node-ok ezen üzenet fogadásának hatására lépnek ki installációs fázisukból, és CI_READ_RESP = 0x79 üzenettel válaszolnak mely titkosítva tartalmazza a mérési adatokat. A válaszüzenet M-, és A-fieldjébe a küldő node M-, és A-fieldje kerül, az adatkoncentrátor ez alapján azonosítja az üzenet küldőjét. Az adatkoncentrátor ezután periodikusan végzi az adatlekérést (15 percenként), mindaddig, amíg a hálózati újraépítés meg nem történik (azaz naponta).

Ügyelnünk kell az üzenetek C-field-jére (*control field*), mely az üzenet módját jelzi. Jelen alkalmazásban három módot használunk, az egyik az installációs üzenet jelzése (mely értéke megegyezik az installációs CI mező 0x06 értékével), 0x04 értékkel a broadcast üzenettípust a *nonce* küldésénél, valamint a 0x08 *request/response* értéket az adatkérés és válaszra.

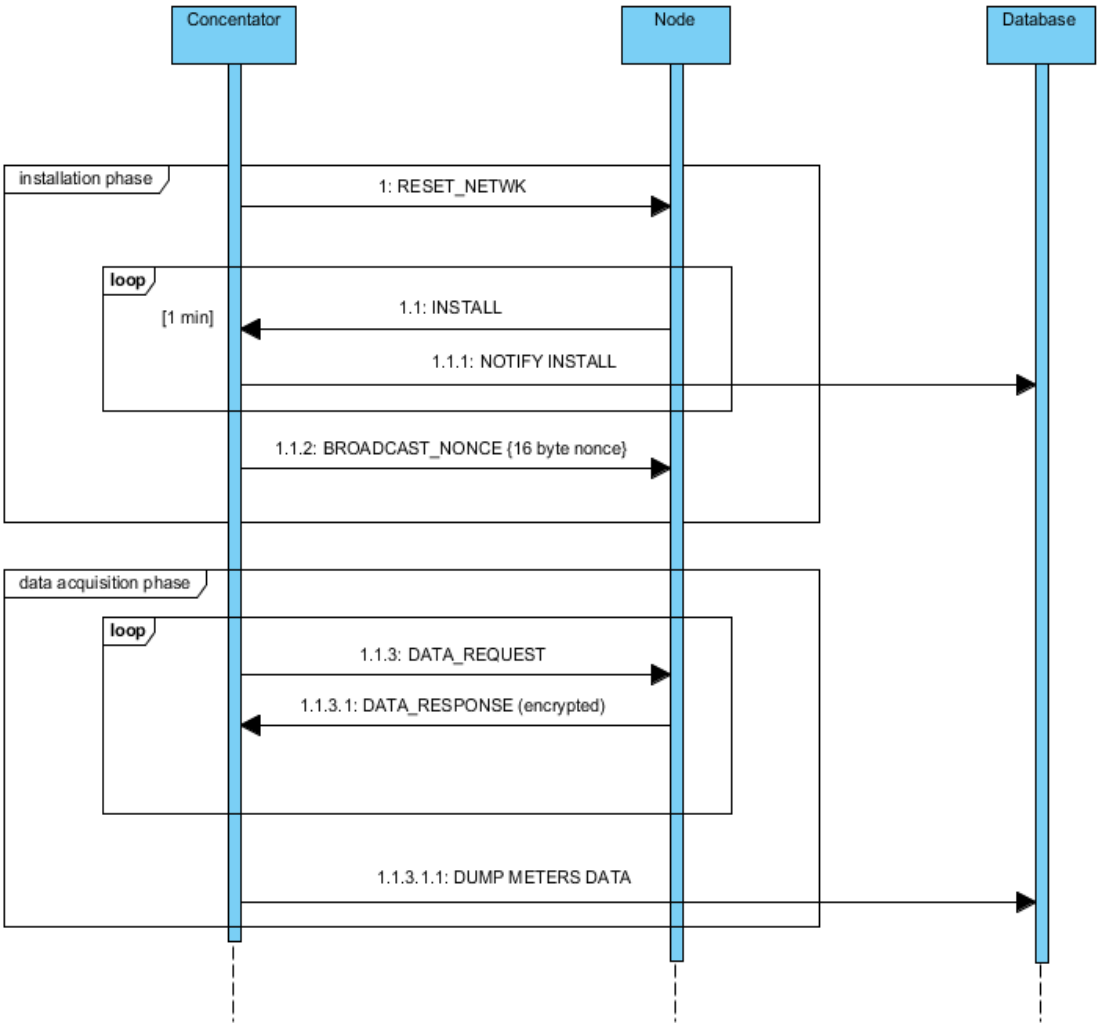
Összefoglalásképp a fent definiált hálózati üzenetek alább kerülnek részletes ábrázolásra.



6.9. ábra. A rádiós adatcsomagok. a) installációs üzenet, b) *nonce*, c) adatlekérés a koncentrátrotól, d) adat válasz a node-tól, e) a hálózat újraépítésének jelzése

A hálózathoz tartozik egy központi szerver is, mely elsődlegesen az adatok tárolását végzi. A koncentrátor egység távoli kapcsolattal rendelkezik ezen szerver felé, jelzi a node-ok installálódását a hálózat monitorozásának érdekében, valamint periodikusan kiírja az összegyűjtött mérőórák adatait.

A hálózat működésének szekvenciája a 6.10. ábrán látható.



6.10. ábra. A hálózat működésének folyamata

6.3. A mérőóra node-ok

A hálózatban a mérőórák mellé telepített egységek továbbítják a mérési adatokat az adatkoncentrátor felé. Az alábbi alfejezetekben ismertetésre kerülnek ezen rendszerelemmel szembeni követelmények, az ezeket kielégítő hardver felépítése, majd a szoftver tervezése és implementációja is. A nyomtatott áramkör megtervezése nem feladata ezen dolgozatnak, így az nem kerül részletezésre.

6.3.1. Követelmények

A mérőóra node-ok funkciója a mérési adatok begyűjtése a mérőóráktól, majd az adatok továbbítása az adatkoncentrátor felé. A kommunikáció 868 MHz vivőfrekvenciájú rádiós kapcsolaton történik, melyet megvalósító elem a 6.1.3. fejezetben került bemutatásra.

A mérőóráktól származó mérési adatok feldolgozásához, valamint a rádiós kommunikáció magas szintű kezeléséhez szükséges egy mikrovezérlő alkalmazása. A választott rádiós egység titkosítása a kulcselosztási algoritmussal együtt nem használható, mivel a kulcsokat a hardverbe inicializáláskor kell megadni. Emiatt a választott mikrovezérlőnek támogatnia kell hardveres, vagy szoftveres titkosítást.

A rendszer méretéből és felhasználásából adódóan szempont lehet a minél alacsonyabb fogyasztás is, így előnyös egy *low-power* mikrokontroller alkalmazása.

Mivel az adatkoncentrátor, és a mérőóra node-ok esetében is ugyanaz a rádiós egység kerül felhasználásra, ezért célszerűnek bizonyulhat a kommunikációs kártya moduláris felépítése is.

6.3.2. A kommunikációs kártya felépítése

A rádiós modul szerepét a már ismertetett Telit ME50-868 tölti be, azonban annak működtetéséhez, valamint a mérési adatok leolvasásához és a kulcselosztási algoritmus implementálásához elengedhetetlen egy mikrovezérlő alkalmazása.

A *low-power* processzorok világában a Texas Instruments MSP430 sorozata igen elterjedt, sok variánsa található meg különböző felhasználási célokra. A választott **MSP430F6779A** mikrokontroller [13] 512KB flash memóriával, valamint 32KB RAM memóriával, több *low-power* móddal, 6 kommunikációs porttal, és AES-128 titkosítást gyorsító hardverrel rendelkezik, ajánlott felhasználási területei között a SmartMeter alkalmazások is szerepelnek.

A választást a processzorcsalád korábbi ismerete, az alacsony fogyasztás, a nagy számú kommunikációs interfész, valamint a hardveres AES titkosítás indokolta. Bár a memória nagy mérete nem indokolt ezen alkalmazásnál, azonban teret ad esetleges további fejlesztéseknek is.

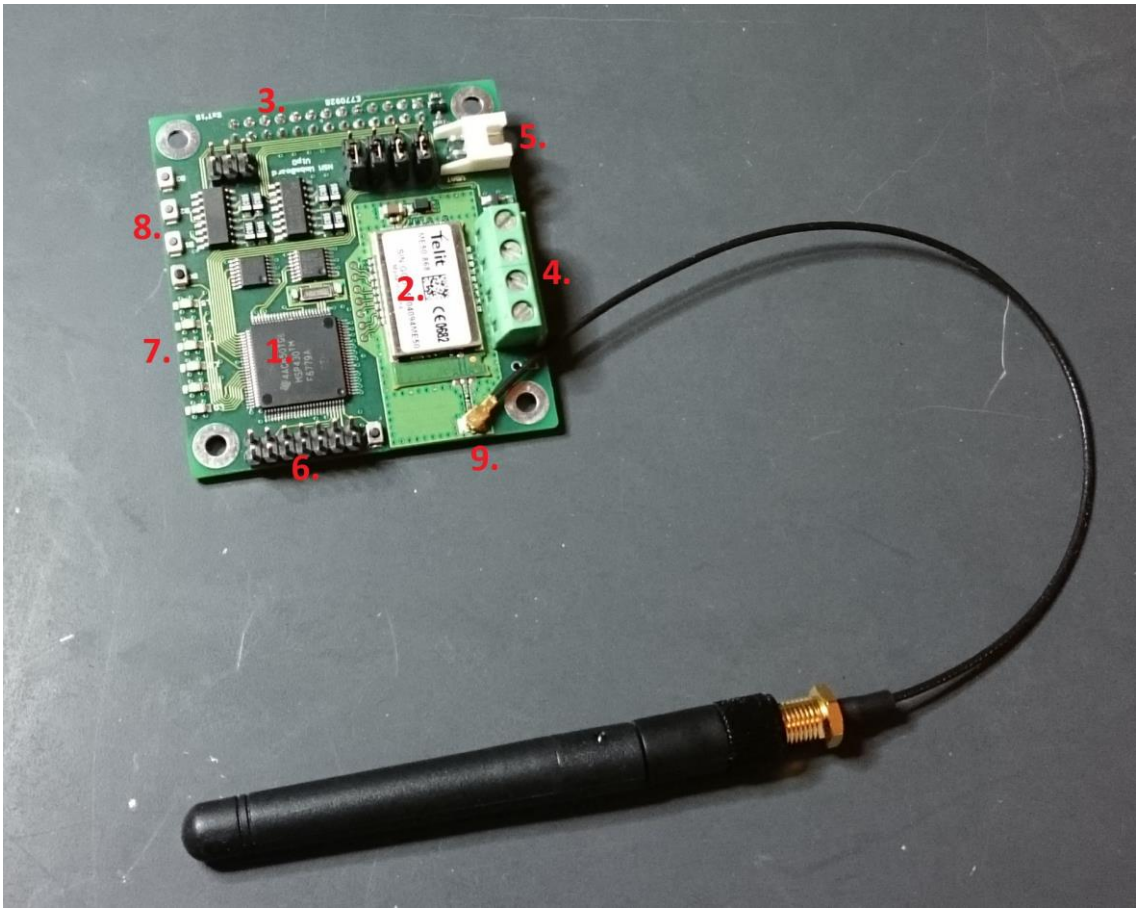
A Telit rádiós modul UART interfésszel csatlakozik a mikrokontrollerhez, az egyéb kommunikációs státuszjelző lábak, mint a CTS (*Clear To Send*) vagy az RTS (*Request To Send*) nem kerültek bekötésre.

A hardver kialakítása moduláris, mely lehetővé teszi a *meter node* felhasználás mellett az adatkoncentrátor hardverbe való beépülését is egy tüskesor csatlakozó segítségével, me-

lyen a gateway applikációs processzor UART csatlakozása található. A modularitást elősegíti a tápválasztó áramkör is, mely lehetőséget biztosít az elemről való működtetésre, vagy a tüskesorról történő megtáplálásra is.

A követelményeknek megfelelően egy külső csatlakozóra került egy további soros port a mérőóra csatlakozásának támogatására. A felhasználó felé történő kommunikáció érdekében 6 LED és 3 nyomógomb, valamint egy reset gomb is került a kártyára.

Az elkészült rádiós kártya [2] az alábbi ábrán látható, a főbb elemei jelölésre kerültek.



6.11. ábra. A rádiós kártya: 1) MSP430F6779A mikrovezérlő, 2) Telit ME-50 rádiós modul, 3) tüskesor csatlakozó, 4) mérőóra csatlakozó, 5) elem csatlakozó, 6) JTAG csatlakozó, 7) felhasználói LED-ek, 8) felhasználói gombok, 9) antenna csatlakozó

A mérőóra node-ok doboza (lásd alábbi kép) tartalmazza a rádiós kártyát, egy 19000 mAh-s elemet, egy kapcsolót, egy kivezetett LED-et, egy nyílást a mérőóra kábelek csatlakoztatásához, valamint egy rádiós antennát.

6.12. ábra. A mérőóra modul doboz nyitott és zárt állapotban

6.3.3. Szoftvertervezés és megvalósítás

A mérőóra node rádiós kártyán futó szoftver az MSP430F6779A mikrokontrolleren kerül megvalósításra. A dolgozat szempontjából elsődleges funkció a rádiós kommunikáció megteremtése, illetve a hálózati applikációs protokoll megvalósítása. A modul 'AT' parancsokkal történő konfigurációja, valamint az adatkeret a 6.1.3. fejezetben részletesen ismertetésre került, a kívánt működéshez szükséges regiszterbeállításokkal együtt, a hálózati applikációs protokoll pedig a 6.2. fejezetben került bemutatásra.

A szoftver tervezésekor törekedünk a megszakítások, valamint a *round-robin* struktúra alkalmazására, oly módon, hogy a megírt függvények nem-blokkoló implementációval rendelkezzenek. Ez azt jelenti, hogy egyes események lefolyása, és várása nem ciklusokban történik, hanem véges állapotgépekben, így felszabadítva a processzoridőt további párhuzamos események feldolgozására is. Előnye a nem-blokkoló függvényimplementációnak továbbá az, hogy egyszerűen blokkolóvá tehető, felhasználástól függően.

A mérőórával történő soros kommunikáció mérőóra hiányában nem került kifejlesztésre, erre későbbi fejlesztés esetén sor kerülhet, azonban a dolgozat szempontjából elegendő, ha a mérési adatokat szimuláljuk.

Az alábbi alfejezetekben a Telit soros kommunikáció, valamint az applikációs réteg kerül bemutatásra.

6.3.3.1. Telit soros kommunikáció

A modullal való kommunikáció soros porton keresztül történik. A választott MSP430 mikroprocesszor négy UART-ként is konfigurálható univerzális soros porttal rendelkezik (USCI_A). A rádiós Telit modul 115200 baudrate-en, 8 bites adatokkal, paritás bit nélkül és egy stop bittel kommunikál, így ezek kerülnek beállításra a periférián. Az USCI_A periféria külön küldési és fogadási shiftregiszterekkel rendelkezik, azonban mélységük egy szó. Az UART meghajtó implementációjakor nem-blokkoló függvényeket, továbbá megszakításokat alkalmazunk.

Az *SendUartWmbus(uint8_t* src, uint16_t len)* függvény egy 'len' hosszúságú buffert küld ki a soros porton, visszatérési értéke UART_TX_DONE = 0x03, ha a buffer elküldésre került, különben UART_TX_BUSY = 0x00. A függvény első hívásakor lemásolja az elküldendő buffert egy átmeneti tárolóba, valamint engedélyezi a küldési megszakítást. A küldési megszakítás engedélyezése után az a kimeneti shiftregiszter üres állapotakor hívódik meg, melyben az átmeneti buffer soron következő byte-ja kerül kiküldésre és számlálásra. A *SendUartWmbus* függvény további hívása során a kiküldött byte-ok számát hasonlítjuk össze az elküldendő buffer méretével. Ha a számláló elérte annak értékét, a függvény letiltja a küldési megszakítások kérését és a visszatérési érték UART_TX_DONE lesz. A küldés aktuális állapota is lekérhető a *TxStateUartWmbus(void)* függvénnyel.

A soros porton való adatbájtok fogadása megszakításban történik, ahol a beérkező byte-ok bufferelésre és számlálásra kerülnek. A *NofCharsRxUartWmbus(void)* függvény visszaadja a buffer jelenlegi méretét, a *ReadUartWmbus(uint8_t* dest, uint16_t* len, uint16_t maxlen)* függvénnyel pedig kiolvashatjuk a buffer tartalmát. Forgó bufferelést alkalmazunk a konkurens hozzáférések elkerülése érdekében, azaz az előbb említett függvények csak pointereken dolgoznak, melyek a két fenntartott buffer-számláló páros közül az épp aktuálisan nem olvasott bufferre és annak számlálójára mutatnak. Így olvasás előtt a pointerek cserélésével biztosítható, hogy a bufferek olvasása közben nem változik azok értéke a megszakításon keresztül.

Ezen függvények adják a soros kommunikáció alacsony szintű kezelését, melyek fölé épül a soros kommunikáció adatkapcsolati rétege. Az AT parancsok küldésére az egyszerű, magas szintű kezelés érdekében létrehozunk különálló függvényeket, mint például a *wmbus_reset_registers(void)* függvényt, mely az ATR<CR> karaktersorozatot küldi el.

Adat módban való küldéskor a 6.2. fejezetben definiált üzeneteket a 6.7. ábrának megfelelően állítjuk össze.

A soros üzenetek fogadását a *wmbus_recv_SM(void)* függvénnyel végezzük el, a működési módnak megfelelően. Konfigurációs módban az AT üzenetek fogadása az alacsony szintű *ReadUartWmbus* függvény segítségével, a beolvasott karakterek bufferelésével történik mindaddig míg <CR> karaktert nem észlelünk. Ekkor a függvény 0x01 értékkel tér vissza, mely után ellenőrizhetjük az üzenet tartalmát. Az egyes AT üzenetek küldése után például blokkolva várhatunk az 'OK' válaszüzenetre, a *wmbus_AT_blocking_wait_for_ok(void)* függvénnyel, mely blokkoló módon (while ciklusban) hívja a *wmbus_recv_SM* függvényt, a ciklus végén pedig összehasonlítja a buffer tartalmát az 'OK' karaktersorozattal.

Adat üzemmódban a *wmbus_recv_SM* függvény feladata a 6.8. ábrának megfelelő adatkeret feldolgozása, az utolsó két mezőt elhagyva. A *ReadUartWmbus* függvény segítségével bufferelejük a beérkezett adatbájtokat, melyeket egy véges állapotgéppel értelmezünk. Elsőként a start karakterre várunk (0xFF), ezt a hossz mező követi, mely után a hossz mezőben specifikált mennyiségű byte következik. Ezután a függvény 0x01 értékkel tér vissza. Ekkor a beérkezett bytesorozatot egy *wmbus_message* üzenet struktúrába mentjük, a beállított Telit adatkeretnek megfelelő bájtokat értelmezve.

```
typedef struct wmbus_message_t
{
	wmbus_c_field c_field;
	uint8_t m_field[2];
	uint8_t a_field[6];
	wmbus_ci_field ci_field;
	uint8_t data_buffer[256];
	uint8_t data_len;
} wmbus_message;
```

Egy sikeres üzenet fogadását egy kéréssel jelezzük az applikációs réteg felé, mely utána feldolgozza azt.

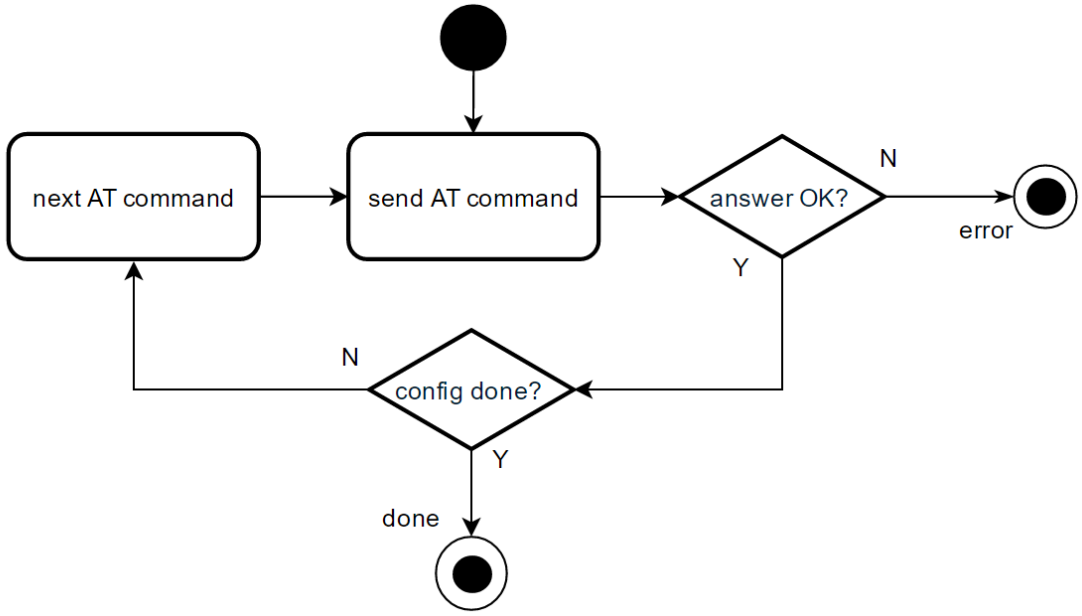
6.3.3.2. Applikációs réteg

A Telit modullal történő kommunikáció bemutatása után az applikációs réteg kerül bemutatásra, mely során megvalósul a hálózati applikációs protokoll.

Mint említésre került, tényleges mérőórával történő soros kommunikáció, és így mérési adat begyűjtés nem került implementálásra, a hálózat működésének demonstrálásához

azonban elegendő az adatok szimulálása. Ezt egy 32 mintából álló 8 bites szinusztáblával érjük el oly módon, hogy időközönként lépünk a szinusz mintákon, és adatkéréskor az épp aktuális értéket küldjük el válaszként.

A program indítása után inicializálásra kerülnek a belső órajelek, valamint a használt perifériák (UART, AES titkosítás, *watchdog*, GPIO, RTC), mely után a Telit modul konfigurálása kezdődik. A folyamat véghezvitelének érdekében egy *wmbus_init(void)* függvényt hozunk létre, mely egy véges állapotgépben (6.13. ábra) küldi el a 6.1.3.1. fejezetben ismertetett AT parancsokat, valamint ellenőrzi a modul által küldött 'OK' válaszokat.

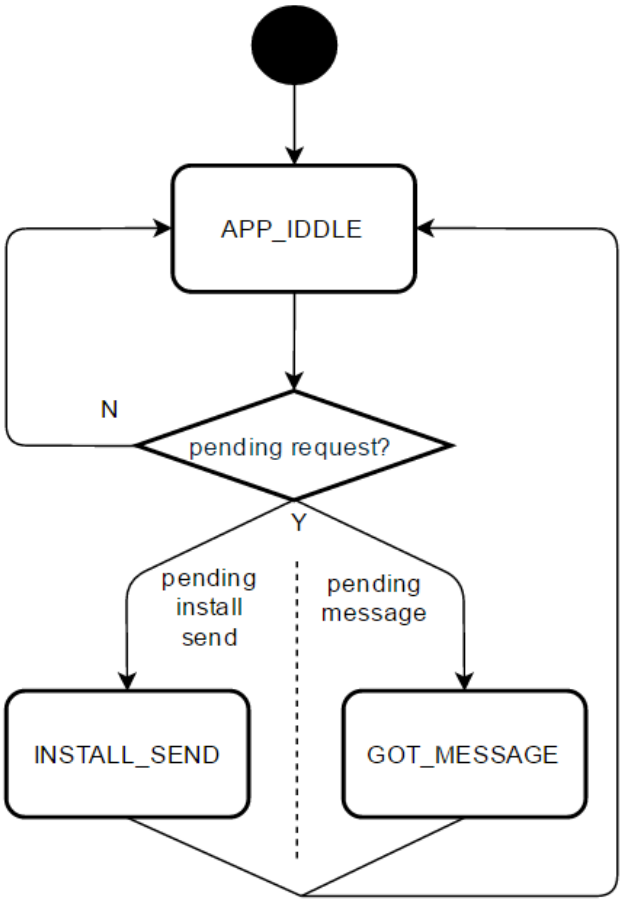


6.13. ábra. Telit konfigurációs állapotgép

A konfiguráció végeztével a modul adat módban üzemel tovább, soros porton továbbítva a fogadott rádiós üzeneteket, valamint rádión továbbítva a soros porton érkező üzeneteket.

A 6.2. fejezetben leírtak alapján a bekapcsolás után a mérőóra modulnak a feladata az installációs üzenet periodikus (percenkénti) küldése. Ezt a mikrokontroller RTC (*Real-Time Clock*) perifériájának segítségével ütemezzük, az installációs üzenet küldésének kérését pedig az applikációs rétegnek egy változóval jelezzük. Ezt csak abban az esetben tesszük meg, ha a *firmware* installációs üzemmódban van, azaz még nem került installálásra az adatkoncentrátor által.

Az applikációs réteg egy állapotgépből áll, mely kéréseket dolgoz fel. Az egyik kérés az előbb említett installációs üzenet küldése, a másik pedig bejövő rádiós üzenet feldolgozása. Ezt szemlélteti az alábbi ábra:



6.14. ábra. Mérőóra applikációs réteg állapotgép

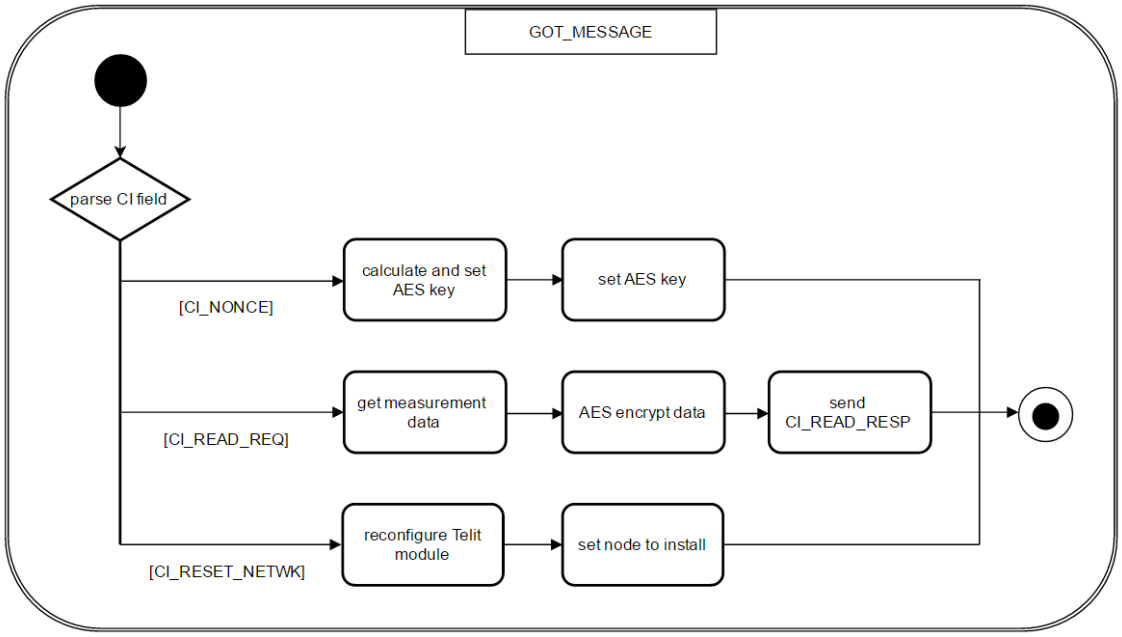
A rádiós üzenetek feldolgozása a CI-field-jük alapján történik. CI_NONCE = 0x77 értékű CI mező esetén az üzenet adatmezője tartalmazza az aktuális titkosítási kulcs kiszámításához szükséges 16 byte-os nonce-ot. A számítás a nonce és az egység egyedi kulcsának XOR kapcsolatából áll elő, mely a mikrokontroller AES titkosítást támogató perifériájába kerül beírásra, mint titkosítási kulcs. Az egység egyedi kulcsa, valamint az M-, és A-fieldje programozáskor kerül beégetésre.

CI_READ_REQ = 0x78 értékű CI mező az adatlekérést jelzi, ennek hatására a szimulált mérési adat titkosításra kerül a korábban beállított titkosítási kulccsal, az AES titkosító hardver segítségével. Ez után CI_READ_RESP = 0x79 üzenetben elküldésre kerül a Telit

modulnak a titkosított mérési adat a 6.2. fejezetben definiált üzenet alapján a 6.3.3.1. fejezetben ismertetett módon. Továbbá ennek hatására a node kilép az installációs módból, azaz nem küld tovább installációs üzeneteket.

CI_RESET_NETWK = 0x80 értékű CI mező a hálózat újraépítését jelzi, mely a node-ot újra installációs módba küldi, továbbá újrainicializálásra kerül a Telit modul, a már ismertetett *wmbus_init* függvénnyel.

Az alábbi ábrán látható összefoglalva az üzenetek feldolgozásának állapotdiagramja:



6.15. ábra. Rádiós üzenetek feldolgozása

Így teljesítésre kerültek a 6.2. fejezetben leírt protokoll mérőóra oldali követelményei, azaz az installációs üzenet periodikus küldése, a nonce üzenet fogadása, az adatok titkosítása és az adatlekérésre történő válasz, valamint az újrainicializáció.

6.4. A gateway egység

A hálózatban a mérési adatok begyűjtését egy *gateway* egység végzi. Ez az egység továbbá távoli kapcsolattal rendelkezik egy szerverrel, mely hosszú távon tárolja az adatokat.

Az alábbi alfejezetekben a *gateway* egységhez tartozó követelmények, az ezek alapján elkészült hardver felépítése, végül pedig a szoftver tervezése és megvalósítása kerülnek bemutatásra.

6.4.1. Követelmények

A *gateway* feladata a rádiós hálózat felépítése, a mérési adatok begyűjtése (adatkoncentrátor), valamint azok továbbítása a központi szerver felé a 6.2. fejezetnek megfelelően. Szükséges, hogy a rádiós kommunikációt megvalósító egység megegyezzen a node-okéval. A szerverrel történő kommunikáció 2G GSM kapcsolaton történik. A feladat specifikációja szerint a *gateway*-t vezérlő szoftvernek Linux környezetben kell futnia, így a vezérlő választásakor megfelelő erőforrással rendelkező egység választása szükséges.

6.4.2. A hardver felépítése

A *gateway* hardver egy interfész kártyából áll, melyen a következő egységek találhatók: egy rádiós kommunikációs kártya (mely megegyezik a mérőóra node rádiós kártya hardverével), egy GSM kommunikációs egységet tartalmazó kártya, valamint az ezeket vezérlő processzoros kártya.

Mint már említésre került, a mérőóra node hardvernél alkalmazott rádiós kártya moduláris felépítésű, a tüskesor csatlakozó lehetővé teszi az interfész kártyához való csatlakozását, melyen keresztül lehetőség nyílik a processzoros kártyával történő soros kommunikációra, valamint a rádiós kártya tápellátására.

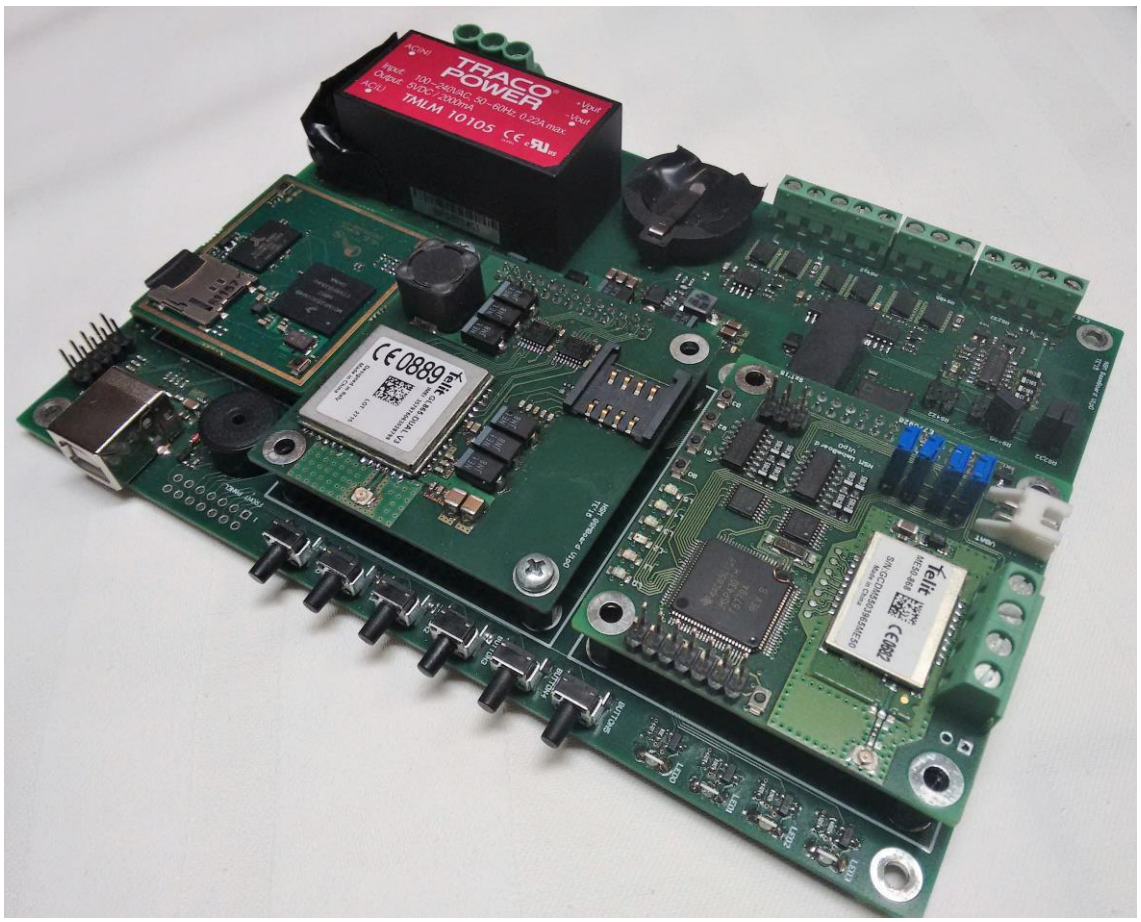
A választott GSM kommunikációs modul a Telit GL865, mely a GSM kommunikációs kártyán került elhelyezésre. A rádiós kommunikációs kártyához hasonlóan, a kártyán található tüskesorra a modul UART kommunikációs vonalai, valamint tápellátáshoz szükséges pinek kerültek kivezetésre. A kártyán továbbá a GSM modulhoz kapcsolódó SIM kártya foglalat is található.

A harmadik interfész modulhoz kapcsolódó kártya a processzoros kártya, melyen egy Freescale i.MX283 ARM9 maggal rendelkező applikációs processzor található. A dolgozat feladatkiírása Linux operációs rendszer alatt futó szoftver fejlesztését írja elő, mely

ezen processzor felhasználásával megvalósulhat. A processzor önmagában nem rendelkezik elegendő RAM memóriával az operációs rendszer támogatásához, így egy hozzá illesztett külső 128 Mbyte-os DDR2 SDRAM is került a kártyára. Az interfész kártya csatlakozóján keresztül kap tápellátást, valamint képes UART periférián kommunikálni a GSM kártyával és a rádiós kártyával. Kivezetésre került továbbá a processzor USB perifériája, melyen keresztül hozzáférhetünk az operációs rendszer termináljához. Található a kártyán továbbá egy SD kártya foglalat, melyen keresztül futtathatjuk az operációs rendszert.

Az interfész kártyán találhatók továbbá LED-ek, valamint az USB csatlakozó a processzoros kártyához. Az interfész kártya tápellátása 220V-os hálózati feszültségről történik, a tápstruktúra ebből állítja elő az egyes kártyákhoz szükséges feszültségszinteket.

Az elkészült hardver az alábbi képen látható [2]:



6.16. ábra. Az elkészült *gateway* hardver egység

A kép bal oldalán a processzoros kártya, középen a GSM modult tartalmazó kártya, míg jobb oldalon a már ismertetett rádiós kártya látható.

6.4.3. Szoftvertervezés és megvalósítás

Mivel a mérőóra node-oknál kifejlesztésre kerültek a Telit modullal való kommunikáció rétegei (UART és adatkapcsolat), ezért kézenfekvő megoldás, hogy a rádiós kommunikációt továbbra is az MSP430 vezérelje, a *gateway* vezérlő (processzoros kártya) mindössze a kártya magas szintű vezérlést végezze soros porton keresztül.

A megvalósítandó szoftver így tehát két elemből áll. Az egyik a *gateway* egységet vezérlő Linux operációs rendszeren futó szoftver (ezentúl *gateway* szoftver), a másik a rádiós kommunikációt kezelő MSP430-on megvalósítandó adatkoncentrátor szoftvere.

Megjegyzendő, hogy a Linux operációs rendszer alatt futó *gateway* szoftver, a processzszoros kártya párhuzamos hardverfejlesztése miatt PC-s környezetben, Ubuntu operációs rendszeren került kifejlesztésre. A soros kommunikáció a rádiós kártyával, valamint a GSM modullal USB-soros átalakítón keresztül valósult meg.

6.4.3.1. Adatkoncentrátor kommunikáció

A *gateway* modul rádiós kommunikációját a mérőóra node-oknál már bemutatott hardver végzi, az MSP430 mikrokontroller által vezérelve. A magas szintű vezérlést, mint a mérési adatok lekérésének jelzése, a nonce generálása és elküldése, valamint a hálózat újraépítésének kérése, a *gateway* szoftver végzi. Ehhez szükséges a két processzor közötti kommunikáció definiálása, valamint a rádiós kártya *firmware* kiegészítése a soros kommunikációval.

Az MSP430 soros port kezelése a 6.3.3.1. fejezetben már bemutatásra került, ennek mintájára implementáljuk a *gateway* soros kommunikációt is az USCI_B perifériáján, melyhez külön megszakítás tartozik. A Telit alacsony szintű kommunikációs függvények mintájára létrehozzuk az *InitUartGateway(void), SendUartGateway(uint8_t* src, uint16_t len),* a *TxStateUartGateway(void),* a *ReadUartGateway(uint8_t* dest, uint16_t* len, uint16_t maxlen),* valamint a *NofCharsRxUartGateway(void)* függvényeket, melyek implementációja megegyezik a már ismertetett függvényekével.

Az adatkapcsolati réteg a következőképp határozza meg az üzenetbájtok jelentését:

START	LENGTH [N]	CMD	DATA x N	CHKSUM	END

6.17. ábra. MSP430 – *gateway* soros üzenetkeret

Az első bájt az üzenet kezdetét jelzi, értéke 0xFF, a második az adatmező hosszát jelzi (lehet 0 is), a harmadik az üzenet típusát, a negyedik bájt a parancs információt adja meg, ezt követi N darab adatbájt. Ezt követi egy, a LENGTH, a CMD, valamint a DATA mezők 8 bites összegéből számított ellenőrző összeg, végül pedig a lezáró <CR> karakter.

Az adatkapcsolati réteg implementációja a Telit kommunikációhoz (lásd 6.3.3.1. fejezet) hasonlóan egy véges állapotgépben valósul meg, mely a *gateway* soros porton beérkező byte-okat dolgozza fel a definiált üzenetkeretnek megfelelően, azaz ellenőrzi a start karakter megérkezését, ellenőrzi a megfelelő mennyiségű adatbájt jelenlétét, valamint az ellenőrzőösszeg helyességét. A lezáró karakter beérkezése és az ellenőrzőösszeg helyessége esetén a függvény 0x01 értékkel tér vissza. Ezután az üzenetkeret megfelelő bájtjait az üzenetmezőknek megfelelő struktúrába parszoljuk, és egy változó segítségével jelezzük a sikeres üzenetfogadást a *gateway* kommunikáció applikációs rétegének.

A *gateway* kommunikáció applikációs rétegének a feladata a beérkezett üzenet feldolgozása és teljesítése, valamint üzenetküldési kérések teljesítése. Az üzenetkeret CMD és DATA mezői itt kerülnek felhasználásra.

A definiált üzenetek az alábbi táblázatban láthatók; zöld háttérrel a *gateway* által küldött üzenetek, kék háttérrel pedig az adatkoncentrátor által küldött üzenetek. Az üzenetek adatmezejének (DATA) jelentése és felosztása függ a parancs mezőtől (CMD).

NAME	CMD	LENGTH	DATA
READ_METER	0x11	24	2 x M_field | 6 x A_field | 8-23: AES128 kulcs
READ_DATA	0x11	14	2 x M_field | 6 x A_field | 4 x DATA
METER_INSTALL	0x20	8	2 x M_field | 6 x A_field
SET_DAILY_NONCE	0x21	16	16 x nonce
RESET_NETWK	0x24	0	-

6.18. ábra. A processzorok közötti kommunikáció üzenetei

A METER_INSTALL üzenetet az adatkoncentrátor küldi a *gateway* felé, jelezve egy installációs üzenet érkeztét. Az adatmezőben az adott *meter* node M-, és A-field-je szerepel.

A SET_DAILY_NONCE tartalmazza az autentikációs kulcsok kiszámításához szükséges *nonce* bytesorozatot, melyet az adatkoncentrátor a már ismertetett CI_NONCE rádiós üzenetben broadcastol a mérőóra node-ok felé.

A READ_METER üzenet adatmezője tartalmazza a leolvasandó mérőóra node M-, és A-field-jeit, valamint a node-hoz tartozó, a nonce-ból és a node titkos egyedi azonosítójából származtatott autentikációs kulcsot. Az adatkoncentrátor ennek az üzenetnek a hatására adatlekérés üzenetet küld az adott mérőóra node felé (CI_READ_REQ), melyre a node CI_READ_RESP rádiós választ küld, mely tartalmazza a titkosított mérési adatokat (lásd 6.3.3.2. fejezet). Az adatkoncentrátor a *gateway* által küldött, a node-hoz tartozó autentikációs kulcs segítségével képes a titkosított adat visszafejtésére, melyet a READ_DATA *gateway* kommunikációs üzenetben továbbít a *gateway* felé.

6.4.3.2. Gateway szoftver

A *gateway* szoftver teremt kapcsolatot a központi szerver, valamint az adatkoncentrátor, és így a rádiós mérőóra-hálózat között, valamint irányítja a hálózati applikációs protokollt. A szoftver Linux operációs rendszeren fut, a fejlesztés Ubuntu operációs rendszeren történik C nyelven, míg a végleges portolt szoftver a processzoros kártyán található Freescale i.MX283 applikációs processzoron fut.

Az alábbi alfejezetekben bemutatásra kerül a többszálú keretrendszer, annak elemei, valamint az applikációs réteg, mely a hálózati applikációs protokollt valósítja meg.

6.4.3.2.1. Keretrendszer

A Linux operációs rendszernek köszönhetően lehetőségünk nyílik több szolgáltatás igénybevételére, mint például a dinamikus memóriafoglalásra, a POSIX thread-ek (szálak) használatara, valamint egyéb kész szoftverelemek használatára (mint azt látni fogjuk a GSM kommunikációnál).

A szoftvert a feladatai alapján három részre tagolhatjuk, melyek külön szálakat kapnak: applikációs szál, GSM kommunikációs szál (*gsm_thread*), rádió kommunikációs szál (*radio_thread*). Az applikációs szál a program fő szála, mely elindítja, valamint karbantartja a többi szálat. Egy szál többek között egy *pthread_t* típusú azonosítóval, valamint egy (végtelen ciklusban futó) függvényre mutató pointerrel rendelkezik, melyeket a szál létrehozásakor meg kell adnunk. A rádiós kommunikációs szál létrehoz egy további szálat a soros porton történő fogadás egyszerű, magas szintű kezelése érdekében.

A szálak egymás közötti kommunikációja az egyszerű kezelés és a rendelkezésre álló erőforrások miatt az operációs rendszer által nyújtott *System V Message Queue* mechanizmus segítségével valósul meg. A *message queue*-k (üzenetsor) láncolt listaként tekint-

hetők a kernel címzési tartományában. Új *message queue*-t az *msgget* függvénnyel hozhatunk létre, melynek első argumentuma egy egyedi kulcs. Ha nem létezik ilyen kulccsal üzenetsor, a függvény visszatér egy azonosítóval, amennyiben már létezik a megadott kulccsal üzenetsor, beállítástól függően annak az azonosítójával, vagy hibaértékkel tér vissza.

Egy üzenetsorba üzeneteket küldhetünk az *msgsnd* függvénnyel, melynek első argumentuma az adott üzenetsorhoz tartozó azonosító, ezt követi az üzenet tartalmára (általában egy struktúrára) mutató pointer. Az üzenet struktúra formájára az egyetlen megkötés, hogy első tagja egy *long* típusú, 0-nál nagyobb értékkel rendelkező változó legyen, mely az üzenet típusát jelöli (*mtype*). Ezt követhetik az üzenet adatváltozói, azonban megengedett adat nélküli üzenet is. A függvény harmadik argumentuma az üzenet méretét adja meg, nem számítva az *mtype* változót. Negyedik argumentumként megadhatunk *flag*eket, melyek meghatározzák a függvény viselkedését teli *message queue* esetén. A függvény sikeres üzenet besorolás esetén 0 értékkel tér vissza.

Egy üzenetsorból üzeneteket vehetünk ki az *msgrcv* függvény segítségével melynek első argumentuma csakugyan az üzenetsorhoz tartozó azonosító. Második argumentuma az üzenetet tárolni képes változóra mutató pointer, a harmadik argumentuma ezen változó mérete. Negyedik változóval megadhatjuk a fogadni kívánt üzenet azonosítóját (*msgtype*).

A keretrendszer lehetőséget ad a program bizonyos paramétereinek konfigurálására egy INI fájl segítségével (*config.ini*). Az INI fájlok szekciókból, és a szekciók alá tartozó kulcs-érték párokból állnak, melyet az alábbi példa INI fájl szemléltet.

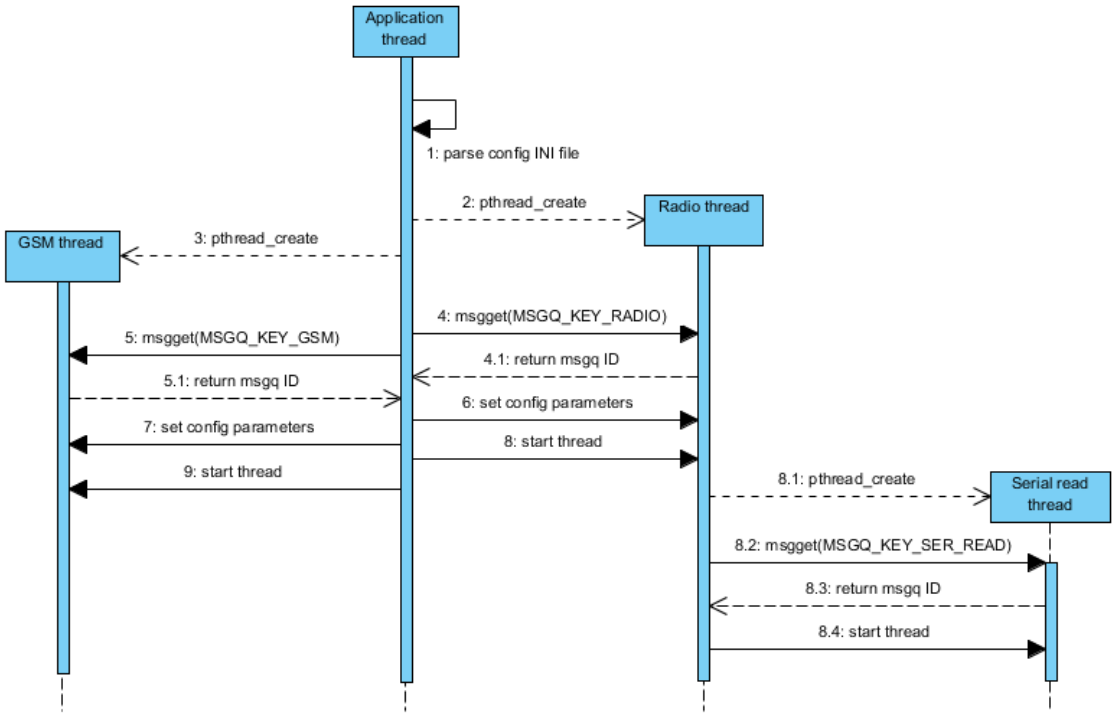
```
[szekcio_1]
pelda_szam=1              ;komment

[szekcio_2]
pelda_string=string       ;komment
```

Külön szekciót hozunk létre az applikáció, a GSM, valamint a rádiós szálak konfigurálása érdekében.

A fájl feldolgozását az applikációs szál végzi egy *open-source* C modul (*inih* [14]) segítségével, a konfigurációs értékek a szekcióhoz tartozó struktúrába kerülnek beolvasásra, majd kerülnek továbbításra.

Az alábbi ábrán látható a keretrendszer felépülése, mely az INI fájl beolvasásával kezdődik, ez után a GSM, valamint a rádiós szálak létrehozása történik, mely során létrejönnek az üzenetsorok és továbbításra kerülnek a konfigurációs struktúrák. A rádiós szál továbbá létrehozza az adatkoncentrátor kommunikáció soros port olvasási szálát.



6.19. ábra. A *gateway* szoftver keretrendszer felépülése

A keretrendszer felépülése után az alábbi alfejezetekben bemutatásra kerülnek az egyes szálak, illetve a definiált szálak közötti üzenetek.

6.4.3.2.2. Rádiós kommunikáció

Mint az már ismertetésre került, a rádiós kommunikációt a rádiós kártyán található adatkoncentrátor végzi, a *gateway* szoftver mindössze magas szintű vezérlést végez az MSP430 processzorral történő kommunikáció során. A két egység közötti kommunikáció érdekében egy egyszerű soros porti protokollt definiáltunk a 6.4.3.1. fejezetben. A keretrendszerben létrehozott rádiós szál célja ennek elrejtése (absztrakciója) az applikációs szál felé.

A már említett config.ini konfigurációs fájl a rádiós szálhoz a következő szekciót tartalmazza:

```
[radio]
serial_device=/dev/ttyUSB1
```

A *serial_device* kulcs segítségével adhatjuk meg az adatkoncentrátorral való soros kommunikáció portját, mely a fejlesztési környezetben az USB-UART átalakító elérési útja.

A szál feladata az applikációs rétegtől, az MSGQ_KEY_RADIO (0x02) kulcsú üzenetsoron érkező üzenetek fogadása, illetve azok hatására a megfelelő soros porti üzenetcsomag összeállítása, valamint elküldése.

A szál külön függvényekkel monitorozza az egyes üzenettípusok (*mtype*) érkeztét az applikációs réteg felől. Ezek az üzenetek a *nonce* elküldésére (MTYPE_RADIO_SET_NONCE), a hálózat újraépítésének kérésére (MTYPE_NETWORK_RESET), illetve mérőóra mérési adatainak lekérésére (MTYPE_RADIO_DATA_REQUEST) vonatkozhatnak. Ezen üzenettípusok érkeztét a már említett *msgrcv* függvénnyel monitorozzák a *__receive_app_set_nonce_message*, *__receive_app_network_reset_message*, valamint a *__receive_app_data_req_message* függvények, melyek argumentumként egy, az adott üzenet struktúrára mutató pointert vesznek át. A mérőóra leolvasás kéréséhez tartozó üzenet például a következőképp alakul:

```
typedef struct  __radio_data_req_message_t
{
	long mtype;
	radio_data_req_message_data_t data;
} radio_data_req_message_t;
```

Ahol a *radio_data_req_message_data_t* az üzenet adat tartalma, mely tartalmazza a leolvasni kívánt mérőóra egység címét (M-, és A-field), valamint a 16 byte-os egyedi titkosítási kulcsát. Hasonlóképp a *nonce* elküldésére vonatkozó üzenet az üzenetsor mechanizmus által elvárt *mtype* változón túl a 16 byte-os *nonce* sorozatot tartalmazza, a hálózat újraépítésére vonatkozó üzenet azonban az *mtype*-on túl nem tartalmaz adatot, mivel arra nincs szükség.

Az applikációs üzenetek sikeres fogadása esetén a függvények 1 értékkel térnek vissza, majd összeállítják az üzenetnek megfelelő soros üzenetet a 6.18. ábrán látható táblázat alapján (zöld sorok), mely után az elküldésre kerül az adatkoncentrátornak.

A rádiós szál létrehoz egy további szálat (*serial_read_thread*) valamint egy hozzá tartozó üzenetsort (MSGQ_KEY_SER_READ = 0x03). A szál feladata a soros porti adatcsomagok fogadása a 6.17. ábrán ismertetett MSP430-gateway üzenetkeret alapján. Egy üzenetbuffer sikeres (*checksum* ellenőrzött) fogadása esetén a soros olvasási szál MTYPE_READ_MESSAGE típusú üzenetet küld a rádiós szálnak, továbbítva az üzenetbuffert.

A rádiós szál a *__receive_serial_message* függvényben nem-blokkoló módon monitorozza az előbb ismertetett üzenetsort az *msgrcv* függvény segítségével, és sikeres fogadás esetén az *__incoming_serial_message_parse* függvényben dolgozza fel azt a CMD mező alapján. METER_INSTALL soros üzenet fogadása esetén értelmezzük az installációs üzenetet küldő mérőóra M-, és A-fieldjét, majd továbbítjuk az applikációs szálnak egy MTYPE_RADIO_INSTALL_MESSAGE típusú üzenetben. A READ_DATA soros üzenet a már dekódolt mérési adatokat, valamint a küldő mérőóra egység M-, és A-field-jét tartalmazza, melyet továbbítunk az applikációs szálnak.

Így megvalósul a rádiós kommunikáció *gateway* applikációs rétegtől való elrejtése, egyszerű magas szintű kezelést lehetővé téve az üzenetsorok segítségével.

6.4.3.2.3. GSM kommunikáció

A rendszerterv bemutatásakor ismertetésre került, hogy követelmény a mérési adatok távoli adatbázisban való tárolása is, amellyel a kapcsolat GSM kommunikáción keresztül valósul meg. A GSM kommunikációt hardveresen a Telit GL865 2G-s egység végzi, mely soros porti interfésszel rendelkezik, és AT parancsokkal irányítható.

Az egységgel történő kommunikáció lebonyolítására lehetőségünk van a PPPD (*Point-to-Point Protocol daemon*) Linux szolgáltatás igénybevételére, mely lehetővé teszi a szabványos AT parancsokkal kommunikáló modem egységekkel való egyszerű kommunikációt.

A PPPD egy önálló Linux szoftvermodul, a kernel alól hívható, és több konfigurációs fájl is tartozik hozzá. Az **/etc/ppp/peers** könyvtár alatt létrehozunk egy *providers* nevű fájlt, mely a modem PPPD beállításait tartalmazza, a fájl tartalma alább látható:

```
/dev/ttyUSB1
115200
nocrtscts
noauth
defaultroute
replacedefaultroute
noipdefault
ipcp-accept-local
nodeflate
persist
connect "/usr/sbin/chat -v -f /etc/ppp/chatscripts/mobile-modem.chat"
```

A fájl első sora megadja a portot amelyen keresztül a GSM modemmel történő kommunikáció megvalósul, utána a baud-rate kerül megadásra, majd a hardveres *flow-control* hiányában az kikapcsolásra kerül. A *defaultroute*, illetve *replacedefaultroute* beállítások azt mondják meg, hogy sikeres GSM csatlakozás után az operációs rendszer alap hálózati interfészként az újonnan létrejött kapcsolatot használja. A *noipdefault* beállítással a lokális IP cím meghatározásának módját befolyásoljuk, mely az *ipcp-accept-local* beállításnak köszönhetően az IPCP protokoll során kerül meghatározásra. A *nodeflate* beállítással kompatibilitási okokból kikapcsoljuk a DEFLATE tömörítési eljárást, a *persist* beállítással pedig azt érhetjük el, hogy a kapcsolat megszakadásakor annak eldobása helyett újracsatlakozással próbálkozzon a program. Végül megadásra kerül egy szöveges fájl, mely a csatlakozás előtti, a GSM modem előkészítéséhez szükséges AT parancsokat, illetve a parancsokra várt válaszokat tartalmazza. Ennek véghezvitelét a *chat* nevezetű Linux program végzi, a fájl tartalma alább látható:

```
'' ATZ
OK "AT+IFC=0,0"
OK "AT+IPR=115200"
OK 'AT+CGDCONT=1,"IP","internet.vodafone.net"
OK "ATDT*99#"
CONNECT
```

A chatscript-ek egy sorban elsőként a modemtől várt karaktersorozatot tartalmazzák, majd szóközzel elválasztva a program által küldött üzenetet. A két aposztróf jelzi az üres sztringet, azaz elsőként nem várunk üzenetre a modultól, mindössze elküldjük az ’ATZ’ parancsot, mely alaphelyzetbe állítja a GSM modemet. ’OK’ válasz esetén az ’AT+IFC=0,0’ paranccsal kikapcsoljuk a forgalomszabályozást, mivel ez hardveresen nem került bekötésre. Ez után az ’AT+IPR=115200’ paranccsal beállítjuk az eszköz soros kommunikációs sebességét. Az 'AT+CGDCONT’ paranccsal az adatcsomag vezérlés

kontextusát állítjuk be, megadva egy azonosítót (1 értékkel), a kapcsolat típusát (IP), illetve a SIM kártya hálózati szolgáltató hozzáférési pontjának nevét (internet.vodafone.net). Ezek beállítása után az ATDT paranccsal hívásra kerül a '*99#' betárcsázási szám, melyre a modem sikeresség esetén a CONNECT karaktersorozattal válaszol.

A *'pppd call provider'* Linux parancsot meghívva az előzőleg ismertetett beállításokkal és szekvenciát alkalmazva kerül felkonfigurálásra a kapcsolat. Sikeresség esetén a Linux bash shell *ifconfig* parancsot futtatva megjelenik a *ppp0* hálózati interfész, mely utána használhatóvá válik, mint internet elérési eszköz.

Egy fontos tervezési lépés még a központi szerver és a *gateway* közötti adatkommunikációs formátum definiálása. Kézenfekvő lehet JSON vagy XML struktúra alkalmazása, azonban léteznek kompaktabb megoldások is, melyek alkalmazása beágyazott eszközökön előnyös lehet a kisebb bonyolultság, vagy a helytakarékosság szempontjából. Egy ilyen tulajdonságokkal rendelkező adatkommunikációs formátum a *MessagePack*. A *MessagePack,* bár hasonlóan kulcs-érték párokat kódol, a JSON-al ellentétben bináris alapú, azaz lehetőséget ad az üzenetek kisebb méretére, illetve gyorsítja az üzenetek feldolgozását, valamint kódolását. A protokollhoz több programnyelven is találunk implementációt, a *gateway* szoftver esetén az *open-source MPack* [15] implementációt alkalmazzuk.

A központi szerver felé jelezzük egy mérőóra hálózati jelenlétét (vagy annak hiányát), a hálózat újraépítését, valamint elküldjük a mérőórák által szolgáltatott mérési adatokat. Ezekhez definiálunk *MessagePack* üzeneteket, melyek felépítése az üzenetsorok üzeneteihez hasonló: az üzenetek rendelkeznek egy *msgtype* mezővel, mely megmondja az üzenet típusát, ezt pedig tetszőleges számú kulcs-érték pár követheti. A mérőóra hálózati jelenlétének jelzésére a létrehozott üzenet tartalmazza a *"msgtype"* mezőt 1 értékkel (ez jelöli a GSM_MSGTYPE_INSTALL üzenetet), az információra vonatkozó mérőóra címét (*address* mező), a mérőóra állapotát (*is_installed* mező) melynek értéke az állapottól függően lehet 0 vagy 1, továbbá a küldés UNIX időbélyegét (*timestamp* mező).

A hálózat újraépítésének jelzésére létrehozott üzenet tartalmazza a *msgtype* mezőt 0 értékkel (GSM_MSGTYPE_RESET), ezen kívül nem tartalmaz más kulcs-érték mezőt.

A mérési adatok küldésére létrehozott üzenet tartalmazza a *msgtype* mezőt 2 értékkel (GSM_MSGTYPE_DATA), a mérőóra node címét az *address* mezőben (M-, és A-field),

a mérési adatot a *value* mezőben, valamint a küldés UNIX időbélyegét a *timestamp* mezőben.

Az így definiált üzenetekre alább láthatunk példákat:

```
// GSM_MSGTYPE_RESET
{
  "msgtype":0
}

// GSM_MSGTYPE_INSTALL
{
  "msgtype":1,
  "address": [1,2,3,4,5,6,7,8],
  "is_installed":1,
  "timestamp":1480805614
}

// GSM_MSGTYPE_DATA
{
  "msgtype":2,
  "address": [1,2,3,4,5,6,7,8],
  "value":1969,
  "timestamp":1480805615
}
```

Ezek az üzenetek kerülnek a *MessagePack* protokoll szerinti kódolásra, majd továbbításra a GSM modemen keresztül a központi szerver felé.

A GSM modemen keresztüli adatküldés a már ismertetett PPPD funkcióval valósul meg, melynek sikeres konfigurációja után rendelkezésünkre áll a ppp0 hálózati interfész. Ennek köszönhetően a szabványos socket kommunikációt annak megnyitásakor hozzákötjük (*bind*), így azon keresztül történhet az üzenetküldés. Ezt szemlélteti az alábbi kódrészlet, mely az ’s’ sockethez rendeli hozzá a ppp0 interfészt:

```
snprintf(ifr.ifr_name, sizeof(ifr.ifr_name), "ppp0");
rc = setsockopt(s, SOL_SOCKET, SO_BINDTODEVICE, (void * ) & ifr, sizeof(ifr));
```

A szerver IP címét és portját a konfigurációs INI fájlban lehet megadni a [gsm] szekció alatt, mely értékek a keretrendszer által továbbításra kerülnek a GSM szálnak:

```
[gsm]
socket_port=8060
server_address=89.104.46.209
```

A keretrendszerben a modemszintű GSM kommunikációt az erre létrehozott szál konfigurálja fel (*pppd* rendszerhívás parancs segítségével), illetve ez a szál végzi az üzenetek küldését (szabványos socket kommunikáció segítségével). A kommunikáció az applikációs szállal az MSGQ_KEY_GSM (0x01) kulcsú üzenetsoron keresztül történik. Jelen implementációban a GSM kommunikáció egyirányú, azaz csak a *gateway* egység küld adatokat a szervernek. Ezáltal az üzenetsor kommunikációja is csak egyirányú, a GSM szál csak fogadja az üzeneteket. A rádiós szálhoz hasonlóan nem-blokkoló módon monitorozzuk az üzenetsort a definiált applikációs üzenetek fogadására.

MTYPE_RADIO_INSTALL_MESSAGE típusú, az üzenetsorból származó üzenet fogadása esetén, az abban található M-, és A-fieldeknek megfelelő GSM_MSGTYPE_INSTALL üzenet kerül elküldésre a socketen "is_installed":1 kulcs-érték párral. MTYPE_RADIO_UNINSTALL_MESSAGE esetén hasonlóan járunk el, azonban "is_installed":0 kulcs-érték párral.

MTYPE_GSM_METER_DATA típusú applikációs üzenet fogadása esetén az üzenetben található M-, és A-fieldek, valamint a mérési adat kerül továbbításra a GSM_MSGTYPE_DATA típusú *MessagePack* üzenetben a socketen keresztül.

A GSM modemen keresztül küldött *MessagePack* üzenetek szerveroldali feldolgozása a 6.5.1. fejezetben kerül ismertetésre.

6.4.3.2.4. Applikációs réteg

Az applikációs réteg feladatai az események vezérlése és időzítése a 6.2. fejezetben leírtak alapján, valamint a *gateway* szoftver magas szintű irányítása a már ismertetett üzenetsorokon keresztül.

Összefoglalva, a 6.2. fejezetben ismertetett protokoll alapján a hálózat periodikusan újraépítést igényel, mely után az installációs üzeneteket csak adott ideig fogadja el a hálózatot vezérlő egység, mely idő leteltével a *nonce* elküldésre kerül. Ezután az installálásra került egységek adatai periodikusan lekérésre kerülnek, melyek egy nagyobb periodicitással a központi szerver felé kerülnek továbbításra. Ezen specifikáció alapján az applikációs rétegnek szüksége van időzítők használatára, valamint a mérőórák nyilvántartására.

Az szoftver időzítései a Linux által nyújtott POSIX időzítővel valósulnak meg. Az időzítők a megadott eseményjelző függvényt hívják meg letelésükkor, melyhez beállítható az első jelzési (*it_value*), valamint a periodikus (*it_interval*) időtartam. Egy processzen belül az összes időzítőhöz egy eseményjelző függvény tartozik, azonban az eseményhez tartozó

információból kinyerhető az előidéző időzítőre mutató pointer (*timer_t**), mely összehasonlítható az általunk létrehozott időzítő pointerével. Egyezés esetén egy, az adott időzítőre vonatkozó *flag* változót állítunk be, jelezvén az applikációnak az idő leteltét. Az applikáció ezeket a változókat monitorozza, melyek hatására elvégzi az eseményhez kívánt funkciót. Az egyes számlálók periódusidejét a már ismertetett *config.ini* fájl [app] szekciója tartalmazza, értékük másodpercben adható meg:

```
[app]
radio_data_acq_time_s=60        ;meter read time
gsm_dump_time_s=900             ;gsm send time
radio_setup_nonce_time_s=900    ;install time
network_reset_time_s=86400      ;network reset time
meter_timeout_s=120             ;meter timeout
```

A mérőórák nyilvántartása egy többrétegű C struktúra segítségével történik, mely tárolja a mérőóra címét (M-, és A-field), installációs állapotát, egyedi titkosítási kulcsát, az aktuális, nonce-ból számított titkosítási kulcsát, a legutóbbi mérési értéket, valamint a mérési érték fogadásának időbélyegét.

```
typedef struct __smartmsn_info_t
{
	bool is_installed;
} smartmsn_info_t;

typedef struct __smartmsn_address_t
{
	uint8_t m_field[2];
	uint8_t a_field[6];
} smartmsn_address_t;

typedef struct __smartmsn_meter_t
{
	smartmsn_info_t info;
	smartmsn_address_t addr;
	uint8_t base_key[16];
	uint8_t encryption_key[16];
	uint32_t value;
	time_t timestamp;
} smartmsn_meter_t;
```

A rendszerhez tartozik egy *meters.cfg* fájl, mely karakteres kódolásban tartalmazza a hálózatban található mérőórák címét (*addr*), valamint a mérőórákhoz tartozó egyedi titkosítási kulcsokat (*base_key*). A fájl sorainak száma határozza meg a lehetséges mérőórák számát. A szoftver indulásakor a fájl beolvasása, feldolgozása, valamint az adatok segítségével a mérőóralista inicializálása a *__parse_meter_list* függvényben történik.

A *meters.cfg* fájl egy sora alább látható:

```
0102030405060708OAFC2BC3094B0601D20056200D23BA0F
```

A karakterek párosával egy byte értéket kódolnak, az első nyolc karakterpár a mérőóra címét, míg a további 16 karakterpár a mérőóra egyedi titkosítási kulcsát adja meg.

A mérőóra struktúrák inicializálása után indul az applikáció, mely feladata elsőként a mérőóra installációs üzenetek regisztrálása. Erre a mérőóráknak a rendszer indulása, illetve a hálózat újraépítése után a konfigurációban megadott ideig van lehetőségük (*radio_setup_nonce_time_s*), az időzítést a *timert_t* típusú *nonce_timer* POSIX számláló végzi. Az installációs üzeneteket a 6.3.2.2. fejezetben ismertetett rádiós szál szolgáltatja az üzenetsoron keresztül. Az üzenet fogadásakor, amennyiben az üzenetben található cím megtalálható a mérőórákat nyilvántartó listában, illetve installációs fázisban van a *gateway*, az adott mérőóra *is_installed* változója 1 értékbe kerül. A mérőóra node installációját a GSM szál üzenetsor MTYPE_GSM_RADIO_INSTALL_MESSAGE segítségével jelezzük az központi szerver felé. A beállított idő letelte után generálásra kerül az autentikációhoz szükséges 16 bájtos *nonce* érték, mely segítségével a mérőórák adott *nonce*-hoz tartozó titkosítási kulcsa is kiszámításra kerül a *nonce*, valamint az egyedi titkosítási kulcs elemeinek XOR kapcsolataként.

```
for(i = 0; i < app.meter_cnt; i++)
{
   for(j = 0; j < 16; j++)
      app.meters[i].encryption_key[j] =
                                        app.meters[i].base_key[j] ^ app.nonce[j];
}
```

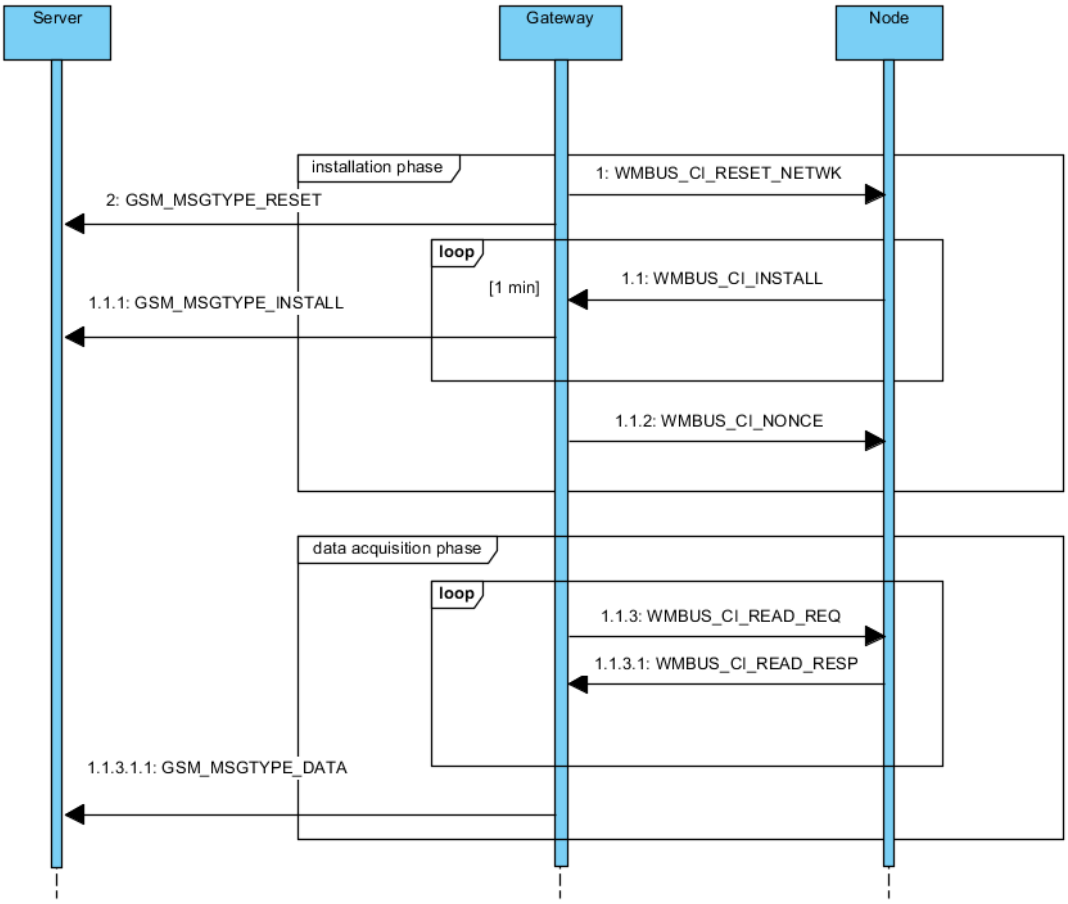
A *nonce* tömb továbbá a rádiós szálon keresztül broadcastolásra kerül a mérőóra nodeok felé. A *nonce* elküldése után elindítjuk az installált mérőórák lekérdezésének ütemezését a *radio_data_acq_time_s* konfigurációs változónak megfelelő periodicitással. Az időzítő letelte után a mérőórák egyesével, egymás után kerülnek lekérdezésre, két lekérdezési üzenet küldése között előre meghatározott időtartamú várakozással. A lekérdezés a már ismertetett rádiós szálon keresztül történik, az üzenetsor MTYPE_RADIO_DATA_REQUEST üzenetével, amely a lekérdezni kívánt mérőóra node címét, valamint a *nonce* segítségével számított *encryption_key* mezőt tartalmazza.

A mérőórák mérési adatait a rádiós szál szoltáltatja az applikációs szál felé, a MTYPE_RADIO_DATA_MESSAGE üzenet segítségével, mely tartalmazza az adott mérőóra címét, valamint a 32 bites mérési adatot. A mérőórához tartozó struktúrában az üzenet fogadásakor a mérési adaton túl későbbi felhasználásra a fogadás időbélyege is elmentésre kerül.

A *nonce* elküldése után a mérési adatok GSM-en történő elküldésének időzítését is elindítjuk a konfigurációs fájl *gsm_dump_time_s* értékének megfelelő periodicitással. Ennek leteltével az installált mérőórák legutolsó mérési adatai kerülnek elküldésre a GSM szál felé egyesével a már ismertetett MTYPE_GSM_METER_DATA üzenetsor-üzenet segítségével. Amennyiben egy mérőóra *timestamp* (ami a legutóbb érkezett mérési adat érkezésekor generált) és az aktuális idő különbségének értéke nagyobb, mint a konfigurációs fájlban megadott *meter_timeout_s* értéke, a mérőórához tartozó *is_installed* változó 0 értékbe kerül, valamint a GSM szál felé a MTYPE_GSM_RADIO_UNINSTALL_MESSAGE kerül elküldésre, jelezvén a szervernek a mérőóra hiányát.

Az applikáció indulásakor a hálózat újraépítésének jelzésére szolgáló időzítő is elindításra kerül a konfigurációs fájl *network_reset_time_s* értékének megfelelő periódussal. Ennek leteltekor az applikáció mind a GSM, mind a rádiós szálnak jelzi a hálózat újraépítését, az MTYPE_NETWORK_RESET, illetve a MTYPE_GSM_NETWK_RESET üzenetekkel. Ezután az applikáció leállítja a mérési adatgyűjtésre, valamint a GSM adatküldésre szolgáló időzítőket, és újra installációs fázisba lép a *nonce* generálására valamint küldésére vonatkozó időzítő elindításával.

A 6.20. ábrán látható az applikáció szekvencia diagramja, jelölve hálózaton küldött üzenetek típusát.



6.20. ábra. Az applikációs réteg szekvencia diagramja, jelölve a hálózaton terjedő üzeneteket

6.5. Szerveroldali adat tárolás, és megjelenítés

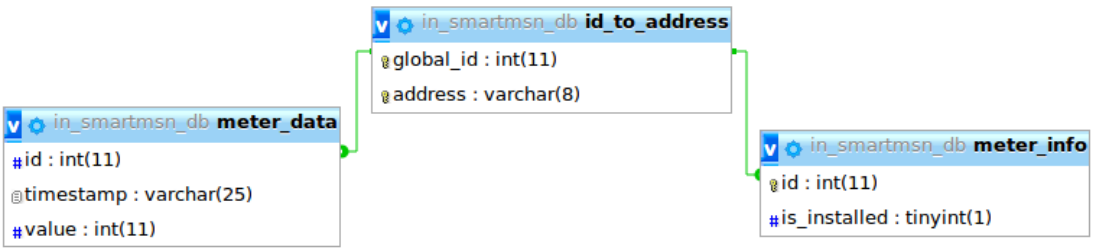
A mérőórák mérési adatai végső formában egy központi szerveren kerülnek tárolásra, mely segítségével azok tovább feldolgozhatók. Jelen alkalmazás esetén a feldolgozás webes grafikus megjelenítést jelent. A szerver egy Ubuntu operációs rendszert futtató PC-n került megvalósításra Python 2.7 környezet, valamint XAMPP keretrendszer segítségével, mely az Apache webszervert, valamint a MySQL adatbázist szolgáltatja.

Az alábbi alfejezetekben a GSM modem által küldött, a 6.4.3.2.3. fejezetben ismertetett *MessagePack* üzenetek feldolgozása, valamint az azokból kinyert adatok adatbázisban történő tárolása, majd az ezek grafikus megjelenítésére létrehozott weboldal kerül bemutatásra.

6.5.1. Adatfeldolgozás, tárolás

Ezen alkalmazásban a szerverprogram szerepét egy Python script tölti be, melynek feladata a 8060-as socket portján beérkező *MessagePack* üzenetek feldolgozása, majd MySQL adatbázisba írása. A *MessagePack* üzenetek az open-source *umsgpack* [16] Python implementációval kerülnek feldolgozásra.

Az adatbázis három táblából épül fel. Az első az *id_to_address* tábla, mely a mérőórák 2+6 byte-os címéhez egy adatbázison belül alkalmazott elsődleges kulcsot rendel hozzá. Az adatbázis többi táblája ezen azonosítón keresztül másodlagos kulcsként hivatkozik az egyes mérőórákra. A következő tábla (*meter_info*) a mérőórához tartozó információkat tárolja, mely jelen alkalmazásban kizárólag a mérőóra installációs státuszát jelenti. A harmadik, és egyben utolsó tábla (*meter_data*) tárolja a mérési adatokat, időbélyeggel ellátva. Az így felépült adatbázis relációs táblája alább látható.



6.21. ábra. A mérőóra adatbázis relációs táblája

A program indításakor kapcsolatot létesít a MySQL adatbázissal, mely sikeressége esetén socket kapcsolaton várja az üzeneteket. Egy üzenet beérkezésekor az *umsgpack* egy Python *dictionary* struktúrába csomagolja ki annak mezőit, szintaktikai helyesség esetén.

Ezután egyszerű módon dolgozható fel az üzenet (az *msgtype* mező értéke például a *pack["msgtype"]* változón kerül érhető el). A feldolgozás az üzenetek már ismertetett *msgtype* mezője alapján történik, mely az üzenetek típusát adja meg.

A *msgtype* 0 értéke esetén a hálózat újraépítése kerül jelzésre, mely az adatbázis szempontjából az összes mérőóra *is_installed* értékének 0 értékbe állítását jelenti, így az ennek megfelelő SQL lekérdezés kerül elküldésre a program által.

```
query = "UPDATE meter_info SET is_installed = 0"
```

A *msgtype* 1 értéke a mérőóra installációs állapotváltozását jelenti, az üzenet tartalmazza a mérőóra címét az *address* mezőben, valamint az installációs állapotot az *is_installed* mezőben. Amennyiben még nem szerepel az *id_to_address* táblában a mérőóra címe, az beillesztésre kerül, valamint az adatbázis automatikusan hozzárendel egy egyedi kulcsot.

```
query = "INSERT IGNORE INTO id_to_address SET address = '{}'".for-
mat(pack["address"])
```

Ezután mindenképpen szerepel az adatbázisban az adott mérőóra, a másodlagos kulcs felhasználásával beírható a mérőóra állapota *meter_info* táblába az alábbi lekérdezéssel.

```
query = "REPLACE INTO meter_info (id,is_installed) VALUES ((SELECT glo-
bal_id FROM id_to_address WHERE address = '{}' LIMIT 1) , {})".for-
mat(pack["address"],pack["is_installed"])
```

A *msgtype* 2 értéke mérési adat érkeztét jelzi, tartalmazza a mérőóra címét, a mérési adatot, valamint egy időbélyeget, melyek beírásra kerülnek az adatbázisba az alábbi lekérdezéssel.

```
query = "INSERT INTO meter_data SET id = (SELECT global_id FROM
id_to_address WHERE address = '{}' LIMIT 1) , timestamp = {}, value =
{}".format(pack["address"],pack["timestamp"],pack["value"])
```

6.5.2. Webes megjelenítés

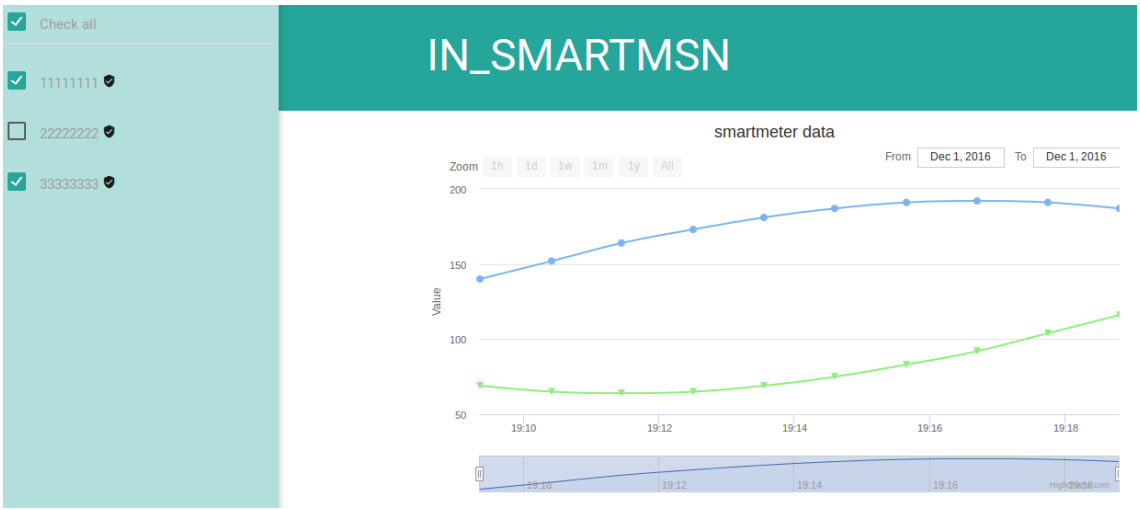
A mérőóra adatok adatbázisba kerülése után azok elérhetővé válnak webes megjelenítés céljából. A felület célja a mérőórák állapotának követése, valamint a mérési adatok grafikus megjelenítése.

Az adatok adatbázisból való lekérdezését *php* technológiával, a front-end megjelenítést pedig *html* és *javascript* technológiákkal valósítjuk meg.

A weboldal egy lapból áll, melynek fő célja a mérési adatok grafikonon történő megjelenítése. A grafikon a *Highcharts*[3] *javascript* modullal kerül megvalósításra, mely egyszerű és gyors felhasználást tesz lehetővé. A grafikon X tengelyén az időbélyegekből képzett dátum és idő szerepel, míg az Y tengelyén a mérési adat. Az időtengelyen mozoghatunk az adatmezőn, vagy a grafikon alatt található sávon egérrel való kijelöléssel, továbbá a grafikon felett található gombok, és beviteli mezők segítségével. A grafikonon egyszerre több mérőóra mérési adata is megjeleníthető.

Az oldal baloldali menüsávja tartalmazza a mérőórák címének listáját, melyek kijelölésével ki-, illetve bekapcsolható a grafikonon történő megjelenítésük. A mérőórák címe mellett található piktogram jelzi az installációs állapotukat.

Az alábbi ábrán látható a weboldal kinézete két mérőóra adatait megjelenítve a kijelölt időtartamon.



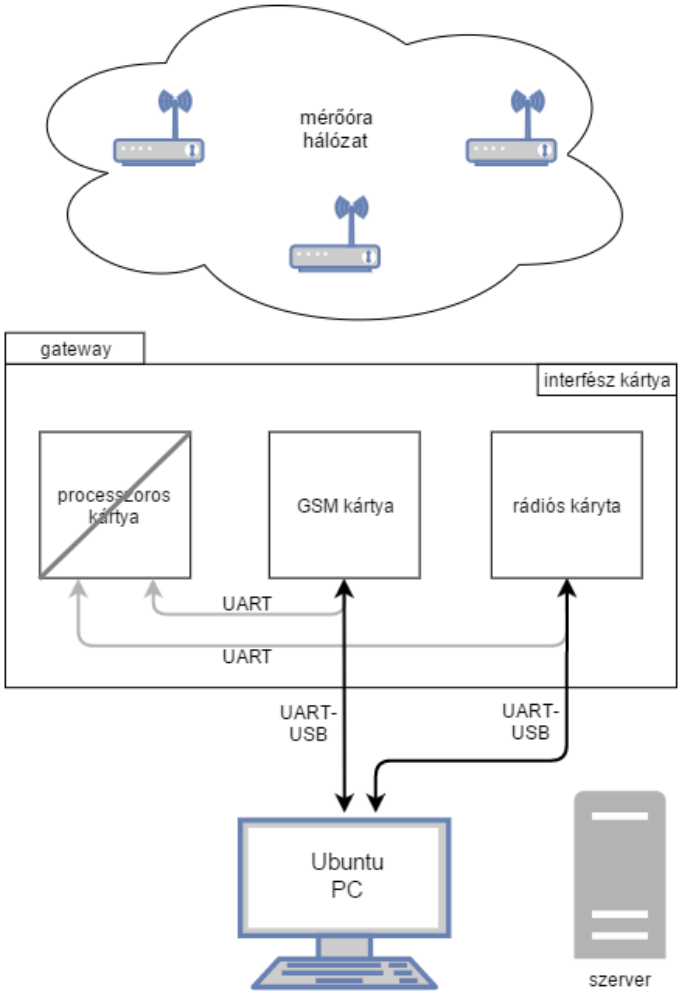
6.22. ábra. A mérőóra adatokat megjelenítő weboldal

[3] Highcharts - http://www.highcharts.com/

7. Demonstrációs alkalmazás

Az előzőekben bemutatott rendszer tervezése és implementációja után sor kerül a működés bemutatására is. A demó három mérőóra node-ból, illetve egy *gateway* egységből áll, melyben a *gateway* szoftver Freescale i.MX283 processzor helyett az Ubuntu operációs rendszerrel rendelkező asztali számítógépen fut, a rádiós, illetve GSM kártyákkal pedig USB-soros átalakítókon keresztül kommunikál.

Erre a demóban azért van szükség, mert a processzoros kártya késői kézhez vétele miatt az arra való szoftver portolásra már nem volt lehetőség. A központi szerver szerepét egy másik asztali számítógép tölti be, melyen az ismertetett Python szerver kód, a MySQL adatbázis, valamint az Apache webszerver fut. A mérési elrendezés az alábbi ábrán látható:



7.1. ábra. A demonstráció mérési elrendezésének sematikus ábrája

A demó célja a rendszer szoftver működésének demonstrálása, azaz a mérőóra-hálózat felépülése, az autentikáció megvalósulása, a mérési adatok kérése és küldése, a hálózat újraépítése, valamint a mérési adatok központi szerver felé történő szolgáltatása.

Mivel a rádiós hálózat tényleges felépítését a Telit rádiós modul végzi, ezért annak működését csak az adatkoncentrátorhoz, illetve általa a *gateway*-hez eljutott üzenetek alapján tudjuk demonstrálni. Emiatt a fő demonstrációs eszköz a *gateway* szoftver konzol kimenete, valamint a webes megjelenítő felület lesz.

A demonstráció a szerver elindításával kezdődik, mely a XAMPP keretrendszer indítását, valamint a Python szerverscript futtatását jelenti. A három mérőóra node egy nagyobb méretű iroda három pontján került elhelyezésre bekapcsolt állapotban, az installációs üzenetüket pedig a leírtak alapján percenként, automatikusan küldik. A mérőóra node-ok szoftveresen beégetett címe az egyszerű emberi azonosítás érdekében sorra egy-egy számjegy sorozatából áll; 1, 2, illetve 3 (lásd a 7.2 ábrán). A *gateway* egység szoftvere ezután kerül elindításra a PC-n. Megjegyzendő, hogy a hálózat működéséhez eddig definiált időtartamok helyett (például napi egyszeri hálózat újraépítés) a demonstráció érdekében rövidebb időtartamok kerültek megadásra a már ismertetett *config.ini* fájlban.

```
[app]
radio_setup_nonce_time_s=90
radio_data_acq_time_s=15
gsm_dump_time_s=60
network_reset_time_s=600
meter_timeout_s=120
```

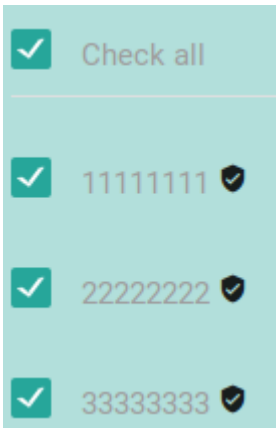
Ez alapján az installációs fázis 90 másodpercig tart, mely után a mérőórák lekérdezése 15 másodpercenként történik, a legutóbbi mérési adatok GSM-en való küldése pedig egy percenként. A hálózat újraépítésére 10 percenként kerül sor, továbbá egy node-ot 2 perc után tekintünk inaktívnak.

Az alábbi képen látható az installációs üzenetek applikációs réteghez való megérkezése. Ez magába foglalja a rádiós üzenetek adatkoncentrátor általi fogadását, az MSP430-gateway közötti soros kommunikáció, valamint a *gateway* szoftver szálak közötti üzenetsor kommunikáció meglétét és sikerességét is.

```
Starting IN_SMARTMSN application.
Starting application.
Creating radio thread.
Creating GSM thread.
Notifying server of network reset!
APP: Installed meter with address [11111111]!
Notifying server of meter install!
APP: Meter with adress [11111111] is already installed!
APP: Installed meter with address [33333333]!
Notifying server of meter install!
APP: Meter with adress [33333333] is already installed!
APP: Meter with adress [33333333] is already installed!
APP: Installed meter with address [22222222]!
Notifying server of meter install!
APP: Meter with adress [22222222] is already installed!
APP: Meter with adress [22222222] is already installed!
```

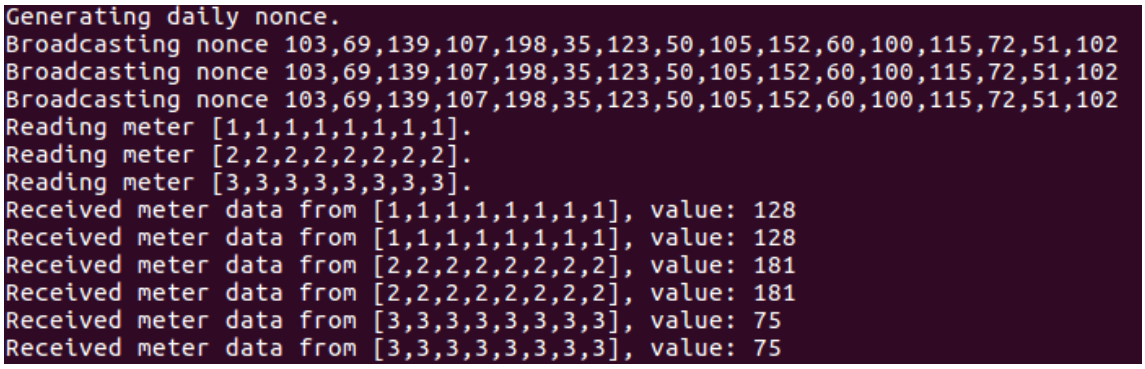
7.2. ábra. Az applikáció indulása és az installációs üzenetek fogadása

Látható, hogy egyes node-ok installációs üzenetei többször is megérkeznek, mivel a hálózatban szereplő többi node által is továbbításra kerülnek. A node-ok belépését a központi szerver felé is jelezzük a PPPD által felépített GSM kapcsolaton keresztül. Ezt a Python script feldolgozása, valamint az adatbázis műveletek elvégzése után a webes felület baloldali menüsávjában a mérőórák címe, valamint a piktogramok megjelenése szemlélteti.



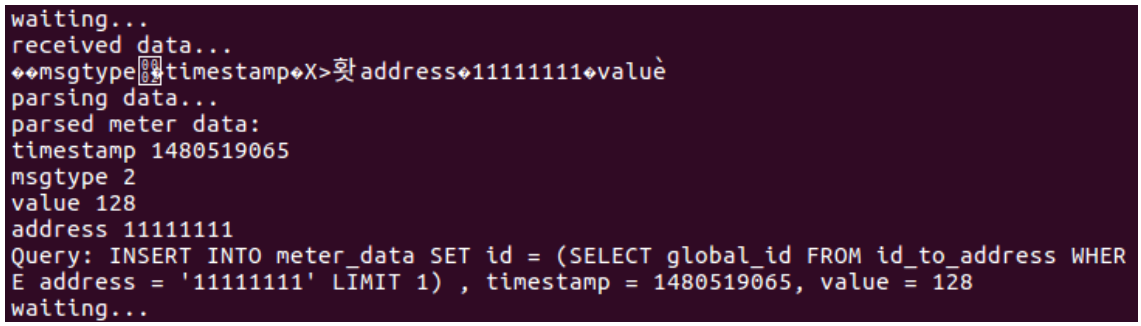
7.3. ábra. A mérőórák állapotának jelzése

Az installációs üzenetek fogadása, valamint feldolgozása után a *gateway* szoftver legenerálja az adott időszakra vonatkozó 16 byte-os *nonce* érték tömböt, melyet utána háromszor broadcastol a rádiós hálózaton. Ezt követően megkezdődik a mérőóra node-ok generált mérési adatainak beolvasása. A *gateway* applikációs réteghez már az adatkoncentrátor által dekódolt mérési adatok jutnak el.



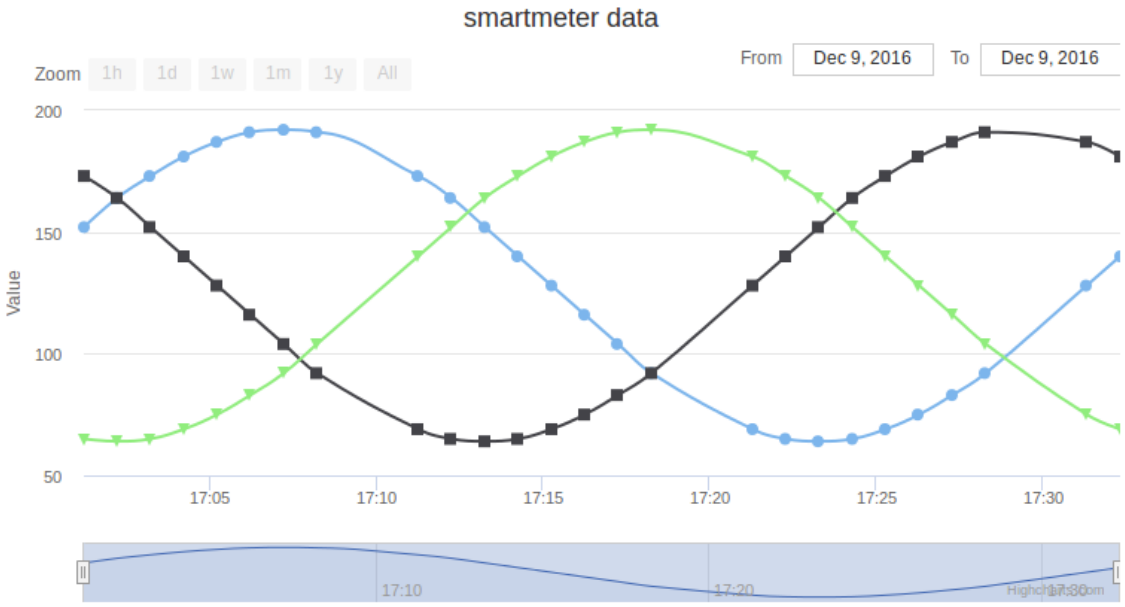
7.4. ábra. A *nonce* generálása, broadcastolása, valamint a mérési adatok lekérdezése

Mindhárom mérőóra node egy 32 mintából álló szinuszjelet generál, percenként új mintát szolgáltatva, egymástól 120°-os fázistolásban. A GSM küldési periódus leteltével az egyes mérőórák legutolsó mérési adata továbbításra kerül a szerver felé, melyek szerver általi feldolgozása a 7.5 ábrán látható.



7.5. ábra. A Python szerverprogram kimenete mérési adat fogadása esetén

Ennek hatására az adatbázisba bekerült mérési adatok megjelennek a webes felület grafikonján. Ez a lekérdezési-szerverküldési ciklus a hálózat újraépítésig periodikusan megismétlődik, mely után újra az installációs üzenetek kerülnek feldolgozásra, és kezdődik elölről a folyamat. Az alábbi ábra több ilyen ciklust szemléltet. A grafikonon a mérési pontok hiánya a 10 percenként történő hálózat újraépítést, és az utána következő installációs fázist mutatja. A 7.6. ábrán látható, hogy a lekért mérési adatok valóban szinuszos jelformát mutatnak, nem tapasztalható kimaradás egyik mérőóra esetében sem.



7.6. ábra. A mérési eredmények

Az applikációs réteghez beérkezett üzenetek, valamint a webes megjelenítés alapján a rendszerelemek közötti kommunikáció, és így a hálózat működőképesnek bizonyul.

Összefoglalás, kitekintés

A jövőben, illetve már napjainkban is érdeke az energiaszolgáltatóknak a mérőórák kézi leolvasás helyetti automatikus adatbegyűjtése, illetve monitorozása. Lehetővé teszi az egyes területek vagy háztartások alaposabb ismeretét, mely alapján a szolgáltató optimálisabb pénzügyi-energiafelhasználási lépéseket tehet meg. Egy, a háztartásoknak is elérhető webes felület a saját energiafelhasználási szokásaikról is gyorsabb visszacsatolású információt adhat a felhasználóknak.

Ezen dolgozatban egy ilyen rendszer tervezése, valamint implementálása került bemutatásra, kezdve a hasonló tulajdonságokkal rendelkező vezetéknélküli szenzorhálózatok területén alkalmazott útkeresési eljárások, valamint a titkosítás tanulmányozásával, ismertetésre került az alkalmazandó wM-Bus mérőóra szabvány, melyet a rendszer elemeinek ismertetése követett a hálózati applikációs protokoll definiálása után.

A hálózati applikációs protokoll a rádiós kártya segítségével került megvalósításra az MSP430 mikrokontrolleren, mely a választott Telit rádiós egységgel valósítja meg a wM-Bus kommunikációt. Ez a kártya mind a mérőóra, mind a gateway modulok esetében felhasználásra került a moduláris felépítésnek köszönhetően. Implementálásra került a gateway Linux operációs rendszeren futó szoftvere is, mely a mérőórák regisztrálását, az autentikáció kezelését, valamint a mérőórák lekérdezését és a mérési adatok GSM-en történő szolgáltatását ütemezi. Végül implementálásra került az adatokat feldolgozó szerver is, mely azokat egy adatbázisba menti, és egy webes felületen megjeleníti.

Az implementált rendszerelemek működése demonstrálásra került, az eredmények kielégítők, azonban teret hagynak további fejlesztéseknek is. Ilyen fejlesztés lehet a gateway szoftver beágyazott eszközre történő portolása, az MSP430 mikrokontroller *low-power* funkcióinak kihasználása, valamint a GSM kommunikáció kétirányúvá tétele, mely a rendszerbe való távoli beavatkozást is lehetővé teheti.

Köszönetnyilvánítás

Szeretném köszönetemet kinyilvánítani a ProDSP Technologies Kft.-nek a diplomamunkám kereteinek megteremtéséért. Köszönöm Molnár Károly, és Orosz György konzulenseimnek a több éves segítséget, Tóth Eszternek a hardver elkészítését, Székely Zoltánnak pedig a lektori segítséget.

Irodalomjegyzék

[1] European standard, *13757-4:2012 "Communication system for meters and remote reading of meters”*, 2012

[2] Tóth Eszter, *Vezeték nélküli okosmérőóra-hálózat hardver elemeinek tervezése*, BME-VIK, MIT, 2016

[3] Jamal N. Al-Karaki Ahmed E. Kamal, *Routing Techniques in Wireless Sensor Networks: A Survey*, Dept. of Electrical and Computer Engineering, Iowa State University, 2004

[4] Rejani. R , Deepu.V. Krishnan, *Study of Symmetric key Cryptography Algorithms*, Research Scholar, Department of Computer Science and Engineering, Manonmanium Sundarnar University, Tirunelveli, 2015

[5] Nechvatal, J., *Public-key cryptography (No. NIST-SP-800-2)*, NATIONAL COMPUTER SYSTEMS LAB GAITHERSBURG MD., 1991

[6] Ajay Kshemkalyani and Mukesh Singhal, *Distributed Computing: Principles, Algorithms, and Systems: Chapter 16: Authentication in Distributed System*, Cambridge University Press

[7] R. Needham and M. Schroeder, *Using encryption for authentication in large networks of computers*, Communications of the ACM, 21(12), December 1978.

[8] Texas Instruments, *Smart Grid & Energy Solutions Guide*, 2015, http://www.ti.com/lit/sl/slym071o/slym071o.pdf

[9] STMicroelectronics, *Spirit1 Low data rate, low power sub-1GHz transceiver datasheet*, 2016, www.st.com/resource/en/datasheet/spirit1.pdf

[10] AMBER wireless, *Compact Wireless M-Bus Radio Module*, https://www.amber-wireless.com/fileadmin//user_upload/DOWNLOADS/amb8426-m_ds.pdf

[11] Radiocrafts, *Wireless M-Bus – Radio module*, https://radiocrafts.com/uploads/rc1180-mbus_features_1_4.pdf

[12] Telit wireless solutions, *Telit Wireless M-Bus Part 4 + Part 5 Mode R2 User Guide, Rev. 6*

[13] Texas Instruments, *MSP430F677xA Polyphase Metering SoCs*, http://www.ti.com/lit/ds/symlink/msp430f6779a.pdf

[14] Ben Hoyt, *inih*, https://github.com/benhoyt/inih

[15] Nicholas Fraser, *MPack*, https://github.com/ludocode

[16] Vanya Sergeev, *umsgpack*, https://github.com/vsergeev/u-msgpack-python

Függelék

1. függelék: AES ECB bemutatása

```
from Crypto.Cipher import AES as AES
from Crypto import Random

key = b'ez egy AES kulcs'
iv = Random.new().read(AES.block_size)
AES_encryptor = AES.new(key, AES.MODE_ECB, iv)

with open('bme_logo.bmp','rb') as im:
    with open('bme_logo_AES_ECB.bmp','wb') as out:
        bmpheader = im.read(14)
        dibheader = im.read(40)
        out.write(bmpheader)
        out.write(dibheader)

        while True:
            chunk = im.read(64*1024)
            if len(chunk) == 0:
                break
            elif len(chunk) % 16 != 0:
                chunk += ' ' * (16 - len(chunk) % 16)

            out.write(AES_encryptor.encrypt(chunk))
```