

Vezetéknélküli érzékelőhálózatok alkalmazása a
katasztrófavédelem területén
c. TDK dolgozat

Készítette: Lőrincz Máté
Mérnök informatikus, 7. félév, BSc

Témavezető: Dr. Oláh András, egyetemi docens



Pázmány Péter Katolikus Egyetem
Információs Technológiai Kar
2012

Tartalomjegyzék

1. Bevezetés és motiváció	5
1.1. Probléma felvetése	5
1.2. Lehetséges megoldások	5
1.3. A rendszer rövid bemutatása	5
1.4. A dolgozat felépítése	6
2. A katasztrófavédelem központi kérdései	8
2.1. Katasztrófaesemények csoportosítása és jellemzői	8
2.2. A túlélők felkutatásának és mentésének módszerei és eszköztára	11
3. Vezeték nélküli érzékelőhálózatok	17
3.1. Történeti bevezető	17
3.2. Protokollok	18
4. Gyakorlatban használt protokollok	20
4.1. ZigBee	20
4.2. Bluetooth	22
5. Hang alapú túlélőkeresés vezeték nélküli érzékelőhálózatokkal	24
5.1. Hangdetekció és lokalizáció	24
5.2. Hangfeldolgozás	26
5.3. Az algoritmusok implementálása WSN-re	31
6. Elosztott számításokat támogató protokoll	36
6.1. Fizikai réteg	36
6.2. Adatkapcsolati réteg	37
6.3. Hálózati réteg	37
6.4. Prezentációs réteg	45
6.5. Alkalmazásréteg	49
7. Mérési és szimulációs eredmények	51
7.1. Hangforrás helyzetének bemérése	52
7.2. Hangfeldolgozás lépései	53
7.3. Protokoll hatékonyságának tesztelése	64

8. Implementáció	67
8.1. A node-ok implementálása	67
8.2. A protokoll implementálása	68
8.3. Mobiltelefon, mint jeladó	72
9. Összefoglalás és a továbbfejlesztés lépései	74

Absztrakt

A felgyorsult technológiai fejlődés egyik központi kérdése a rövid- és hosszú távú kockázatminimalizálás. A biztonsági szempontok már a mérnöki vívmányok tervezés során érvényre jutnak, hogy a balesetek bekövetkezésének valószínűsége minimalizálható legyen, ugyanakkor a természeti katasztrófák után általában csak kármentésre nyílik lehetőség. Ráadásul a civilizáció fejlődése nyomán a védendő erőforrások nagyvárosokba sűrűsödnek, ahol súlyos anyagi és emberi életekben megnyilvánuló károk keletkeznek.

A kurrens infokommunikációs technológiai megoldások sokasága ültethető át a katasztrófavédelmi gyakorlatba, melyek közül a dolgozat fókuszában a vezeték nélküli érzékelő hálózatok alkalmazhatósága áll. A vezeték nélküli kommunikáció és a digitális elektronika fejlődése lehetővé tette az olcsó, multifunkcionális node-ok megjelenését, amelyek a hagyományos érzékelő berendezéseknél lényegesen kisebbek és képesek a rádiós kommunikációra. Ennek a technológiának a segítségével jöttek létre a vezeték nélküli szenzorhálózatok.

A TDK dolgozat középpontjában egy olyan, túlélők utáni keresés során felhasználható alkalmazás áll, amelynek alapja egy, az épületben elhelyezett, inaktív szenzorhálózat, mely egy katasztrófa bekövetkezése után aktiválódik, a csomópontokon lévő mikrofonnal pedig a felhasználó okostelefonján futó alkalmazás hangmintázatát keresi. A kutatócsoportok mozgó adatgyűjtőjéhez az egyes node-ok egy valószínűségi mértéket továbbítanak, melynek segítségével lokalizálhatóak a csapdába esett túlélők. Annak érdekében, hogy a megfelelő szenzorsűrűség biztosított legyen, a hálózat (gazdasági okokból) csak nagyon olcsó szenzor node-okkal valósítható meg. Mivel az olcsó csomópontok a feladathoz nem rendelkeznek elegendő számítási kapacitással, szükséges, hogy a számításokat egy kijelölt klaszteren belül egymás között elosszák. A dolgozat bemutatja az alkalmazáshoz kifejlesztett protokollokat, az eredményeket validáló szimulációs eredményeket, valamint az alkalmazás implementációs tervét, és a főbb részeinek implementációjából létrehozott kísérleti prototípust.

Abstract

One of the main problem of the accelerated technological development is the short- and long-term risk minimalization, which can be traced in every engineering field. The safety aspects of the engineering design process have been enforced, because the engineers have to reduce the chances of the accidents or disasters. Unfortunately, there are limited opportunities for natural disasters (earthquake, tsunami, etc...). The current methods focus on the forecasting and on the rescue. The local natural disasters are dangerous, because during the development of the civilization, the resources – which can simultaneously destroy – are concentrated in larger cities. This damage can be measured in human lives.

Lots of current information and communication technology solutions can be used during a disaster. This dissertation focuses on the wireless sensor networks. The development of wireless technologies made it possible to create high performance, low cost, multifunctional sensor nodes, which are smaller, than the conventional detection systems, and able to communicate by radio. These led to the birth of the WSN systems, which consist many nodes scattered on a large area. The self-organization, the distributed computing and the sensing are fundamental properties of this WSN systems.

This dissertation is focused on a WSN, which is in a building in inactive state. After a disaster, this system goes to active state and begin the regular operation. During this, the nodes are trying to find the survivors, who have a telephone. This telephone detects the disaster, too, and plays a pre-recorded voice, which can be detected by the nodes. This nodes have to be very cheap (because economic reasons), but for this application, they need lots of memory and computing power. They can share the resources with each other, but for this, they need a special protocol, which supports this. In this dissertation, the demonstration of such a protocol will be discussed.

1. Bevezetés és motiváció

1.1. Probléma felvetése

Az elmúlt évszázadok során a katasztrófák előrejelzése egyre fontosabb kutatási területté vált. Ennek oka, hogy a népességnövekedés miatt a veszélyes övezetekben is (törésvonalak mentén, vulkánok környékén) nagy népességű városok alakultak ki. Sajnos, az elmúlt évek tragikus eseményei bizonyítják, hogy még mindig nem áll rendelkezésre olyan technológia, amivel időben fel lehetne készülni a különféle katasztrófákra, nem beszélve a civil lakosságot sújtó célzott támadásokról (terrorizmus, háborúk).

Egy katasztrófa egész városokat semmisíthet meg. Ilyenkor a legfontosabb feladat a romok alatt rekedt túlélők mielőbbi felkutatása és kimentése. Ez egy nagyon aktuális terület, rengeteg kutatás és államilag támogatott projekt célja egy olyan rendszer kifejlesztése, amivel minimalizálni lehet a keresési időt [24]. Jelenleg nem áll rendelkezésre olyan technológia, ami képes hatékonyan és automatizáltan detektálni a túlélőket. Jelen dolgozat egy ilyen rendszer megtervezését és bemutatását tűzte ki célul.

1.2. Lehetséges megoldások

A túlélők felkutatására számtalan alternatív megoldás kínálkozik, amelyeknek általában közös jellemzője, hogy a kutatást végző személy a veszélyes területre behatolva különböző, véletlenszerű helyeken bonyolult és lassú méréseket végez el. Ez a mérés történhet kutyával, mikrofonnal, radarral vagy bármilyen más módszerrel, de a mérés során a mentőegységek folyamatos veszélyben vannak. (A romok nagyon instabilak, bármikor tovább omlhatnak.) Éppen ezért lenne fontos, hogy a mérések automatikusak legyenek és ne igényeljenek emberi beavatkozást. Ilyen jellegű mérésekre a legideálisabbak a WSN-alapú rendszerek.

A szenzorok kiválasztása is fontos kérdés. Lehet mérni egyrészt az emberi test által kibocsátott kémiai anyagokat, ezzel viszont nagyon nehéz meghatározni a pontos helyzetet, ráadásul jelenleg nem léteznek elég érzékeny kémiai szenzorok sem. A túlélőket lehet keresni hangok alapján is. Léteznek olyan mikrofonok, amik olcsók, érzékenyek és könnyen beépíthetők egy kis méretű csomópontba. A hangmintázatok detekciójának nehézsége, hogy a továbbításuk nagy sávszélességet, a feldolgozásuk pedig nagy memóriát igényel. Ezek közül egyik férhető hozzá egy tipikus WSN-hálózat esetén.

A nehéz feldolgozása ellenére, a hang mérése járna a legtöbb eredménnyel, hiszen így a helymeghatározás is elvégezhető. A dolgozatban bemutatott WSN-rendszer éppen ezért a mobiltelefonok által kibocsátott hangot dolgozza fel.

1.3. A rendszer rövid bemutatása

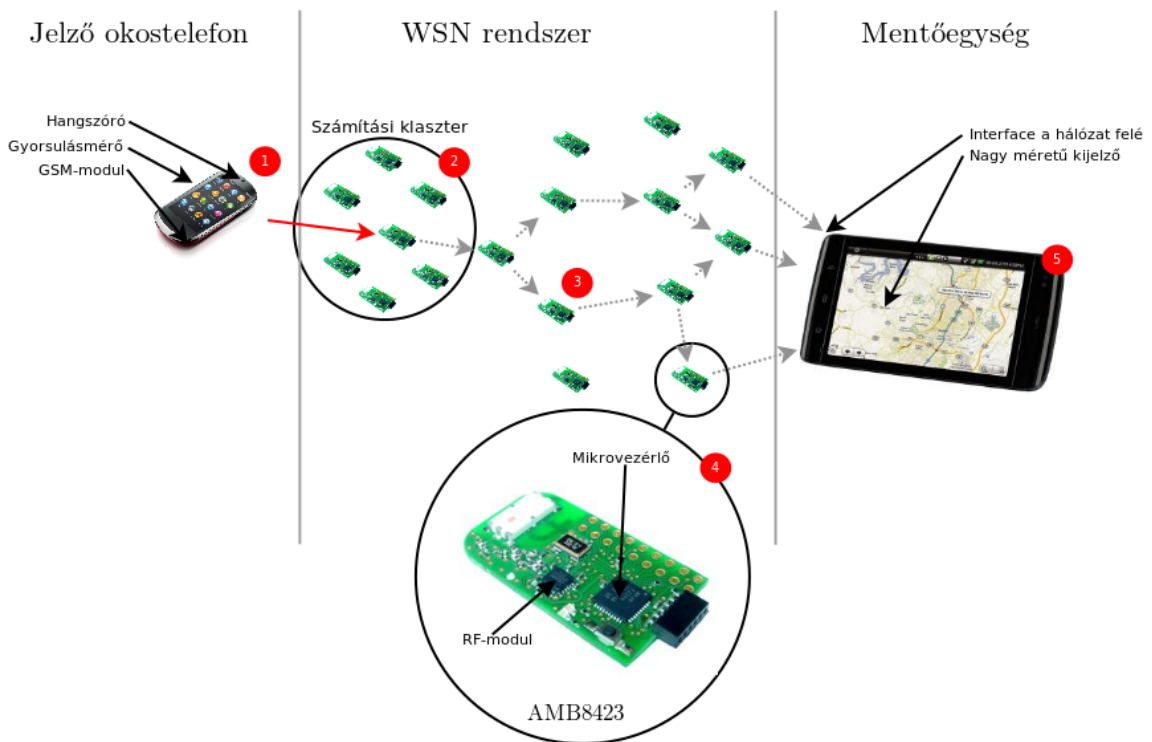
Az 1. ábrán látható a rendszer vázlatos felépítése. Az ábrán jelölt (1)-es pont egy megfelelően felprogramozott okostelefont mutat, ami a beépített GSM-modullal és gyorsulásmérőkkel folyamatosan figyeli környezetét. Ha katasztrófára utaló jeleket érzékel (hirtelen gyorsulás, térerősség megszűnése, környező WiFi-kapcsolatok megszűnése, stb...) elkezd lejátszani egy előre rögzített hangmintát.

A (4)-es pont egy csomópontot mutat be. Egy ilyen eszköz tartalmaz egy olcsó mikrovezérlőt és egy alacsony fogyasztású RF-modult. A csomópontok nem tudják továbbítani

a nyers hangmintát (túl alacsony hozzá a hálózat áteresztőképessége) ezért kénytelenek azt feldolgozni. Mivel ehhez nincs elég memória és számítási kapacitás az egyes modulokban, össze kell dolgozniuk. Az elosztott jelfeldolgozás a (2)-es pontnál látható számítási klaszter segítségével történik. A fizikailag egymáshoz közel álló csomópontok a dolgozat során ismertetett protokoll segítségével képesek megosztani egymás között a memóriát és a számítási kapacitást.

Miután a klaszterben megtörténik az adatok feldolgozása, a kiértékelt eredményt el kell juttatni a nyelőhöz. Erről egy másik, a dolgozat során szintén ismertetett protokoll (multipath ring routing protokoll) gondoskodik. A képen látható szürke szaggatott nyilak (3) jelképezik azt, hogy a klasztertől a nyelőig nem a teljes adatfolyam, csak a feldolgozott, lényeges információk jutnak el.

A mentőegységnél lévő nyelő (5) egy megfelelően felprogramozott tablet ami képes fogadni a hálózat üzeneteit és vizuálisan megjeleníteni az eredményeket. A tablet által szolgáltatott információk segítségével a mentőegységek célirányosan kereshetik a túlélőket.



1. ábra. A rendszer vázlatos felépítése

1.4. A dolgozat felépítése

A dolgozat felépítése során az volt a célom, hogy bemutassam a folyamatot, ami a rendszer megtervezéséhez vezetett:

- a második fejezetben (**A katasztrófavédelem központi kérdései**) a motivációk, az eddig elért eredmények, valamint a friss kutatási területek kerülnek bemutatásra;
- a harmadik és negyedik fejezet (**Vezeték nélküli érzékelőhálózatok és Gyakorlatban használt protokollok**) mutatja be a WSN-rendszerek kialakulását, kihívásait, kutatási területeit és a jelenleg elterjedt protokollokat;

- az ötödik fejezet (**Hang alapú túlélőkeresés vezeték nélküli érzékelőhálózatokkal**) ismerteti a hang alapú helymeghatározást és bemutat egy egyszerű detektáló algoritmust, ami képes megtalálni egy speciális hangmintázatot;
- a hatodik fejezet (**Párhuzamos számításokat támogató algoritmusok**) ismerteti egy olyan protokollt, ami lehetőséget ad arra, hogy a WSN-ben lévő olcsó csomópontok összedolgozzanak egymással, megnövelve ezzel a memória- és számítási kapacitásukat. (A megnövelt kapacitással lehetőség nyílik olyan mérések elvégzésére is, ami klasszikus WSN esetén megoldhatatlan lenne);
- a hetedik fejezetben (**Mérési és szimulációs eredmények**) összegyűjtve helyet kapnak azok a mérések és szimulációk, amik az előző két fejezet megállapításait alátámasztják;
- végül a nyolcadik fejezet (**Implementáció**) bemutatja a protokoll néhány részének egy lehetséges implementációját.

2. A katasztrófavédelem központi kérdései

A természeti katasztrófa olyan súlyos természeti csapást jelent, ami sok embert, gyakran egész városokat vagy országokat érint. Nagyon változatos események sorolhatóak ebbe a kategóriába. Ide tartozik az aszály, mely az emberiség történelme folyamán több uralkodó és birodalom vesztét okozta, a földrengés, mely minden évben milliárdokban és emberéletek százaiban mérhető károkat okoz, az árvíz, ami a modern és folyamatosan fejlesztett védelem ellenére minden évben lakhatatlanná tesz több folyó menti települést, az avartűz, ami nyaranta egész településeket zár el a külvilágtól és a tornádók, amik ellen az USA sem tud védekezni. Nem beszélve a valamivel ritkább, de annál súlyosabb vulkánkitörésekről, tsunamikról és az ember által okozott katasztrófákról, mint például ciánszennyezés, iszapömlés vagy nukleáris katasztrófa.

A természeti katasztrófák nagyon sokféleképpen jelentkezhetnek, de közös bennük, hogy nem lehet őket megakadályozni, legfeljebb az okozott károkat mérsékelni. A Föld minden pontján, minden régióban vannak olyan hivatalos vagy civil szervezetek, amik elsődleges feladata a katasztrófák utáni gyors reakció és a mentési munkálatok azonnali megkezdése.

Amerikában ez a szervezet a Centers of Disease Control and Prevention. 1946-ban, Atlantában alakult. Azóta több vészhelyzet elhárításában is segédkezett: megállította a maláriát, kutatta a veszettség terjedését, kiemelkedő szerepet játszott a himlő felszámolásában. A CDC lassan átvette a katasztrófavédelem néhány feladatát is.

Az Asian Disaster Reduction Center (ADRC) egy nemzetközi szervezet, melynek 29 ázsiai ország a tagja. Az ADRC 1998-ban alakult azzal a céllal, hogy növelje a katasztrófavédelmet a tagországokban, biztonságos közösségeket hozzon létre és hogy megalapozzon egy olyan társadalmat, melyben lehetséges a fenntartható fejlődés. Az ADRC rengeteg nemzetközi programot szervez, melyek során a tagországok kicserélhetik egymással tapasztalataikat.

Magyarországon az Országos Katasztrófavédelmi Főigazgatóság koordinálja a katasztrófavédelmet, melynek két fő feladata a megelőzés és a kárminimalizálás. Munkája során együttműködik a helyi civil szervezetekkel és a tűzoltósággal.

2.1. Katasztrófaesemények csoportosítása és jellemzői

A dolgozat célja egy olyan rendszer bemutatása, mely egy katasztrófa után segít megtalálni a romok alatt rekedt túlélőket. Egy ilyen rendszer megtervezéséhez először meg kell vizsgálni az épületomlások pontos okait és a mentés lépéseit. Ez a fejezet az okokról szól. Végig kell gondolni, hogy milyen események vezethetnek egy épület összeomlásához. Erre azért van szükség, mert a rendszernek az omlás után is működőképesnek kell maradnia, ehhez viszont pontosan ismerni kell a fellépő körülményeket.

2.1.1. Földrengés

A legnagyobb károkat világszerte egyértelműen a földrengések okozzák. Az 1. táblázat az öt legtöbb emberéletet követelő földrengést mutatja az elmúlt 50 évből.

1. táblázat. Az elmúlt 50 év legsúlyosabb földrengései [1]

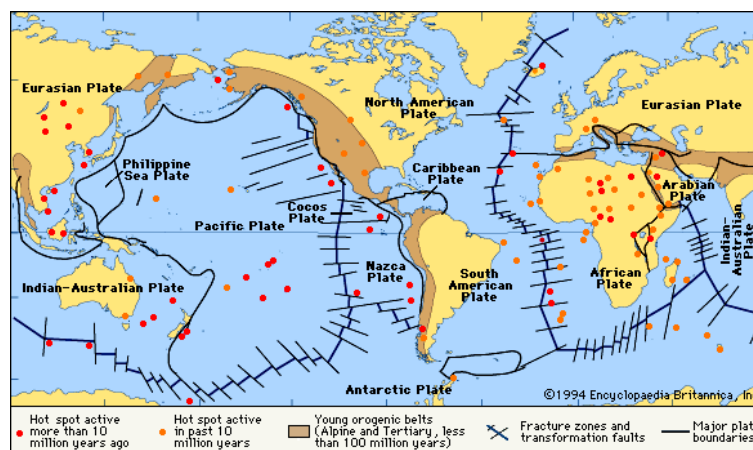
Ország	Dátum	Áldozatok száma
Kína	1976. 07. 27	242000
Haiti	2010. 01. 12	222570
Indonézia	2004. 12. 26	165708
Kína	2008. 05. 12	87476
Pakisztán	2005. 10. 8	73338

A földrengések erejét Richter-skálán mérik. Értéke a fészekben felszabaduló energia logaritmusával arányos. A skála nem lineáris. Egy 5.4-es és egy 6.4-es rengés energiája között a különbség például harminckétszeres. A Richter-skála értékeinek jelentése és gyakorisága a 2. táblázatban látható.

2. táblázat. Földrengések jellemzése erősségük alapján

Richter	A pusztítás mértéke	Ilyen rengések gyakorisága
<2.0	csak műszerekkel érzékelhető	naponta 8000
2.0-2.9	a legtöbb ember nem érzi	naponta 1000
3.0-3.9	érezhető, de nem okoz károkat	évente 49000
4.0-4.9	ritkán okoz károkat	évente 6200
5.0-5.9	a gyenge épületekben rongálódás	évente 800
6.0-6.9	Az erős épületek is rongálódhatnak	évente 120
7.0-7.9	súlyos károk, épületek összeomlanak	évente 18
8.0-8.9	nagyon súlyos károk, hegyomlások	évente 1
9.0-9.9	rendkívüli pusztítás, a táj megváltozik	20 évente 1
≥10.0	a földkéreg kettérepedne	nem történhet meg

Látható, hogy évente 19-20 esetben van olyan súlyos földrengés, ami akár egész városrészeket is összedönthet. A földrengések a földkéreg törésvonalai mentén a leggyakoribbak, így a veszélyeztetett helyeken lévő országoknak számítaniuk kell rájuk. A 2. ábrán a törésvonalak láthatóak.



2. ábra. Térkép, melyen a nagyobb törésvonalak láthatóak

Rengéskárok esetén általában nem egy épület dől össze, hanem egy egész városrész. A környéken megszűnik az áramellátás és a mobiltelefon-szolgáltatás is, amit figyelembe kell venni a mentést segítő eszközök megtervezésekor. Ha a rendszer egy kritikus komponense függ a mobilszolgáltatótól, a hálózati áramellátástól vagy bármilyen, előre kiépített infrastruktúrától, akkor egy földrengés esetén az egész rendszer használhatatlan lesz. A GSM, 3G, 4G és WiFi alapú szolgáltatásokra épülő megoldásokat tehát el kell vetni.

2.1.2. Áradás és hurrikánok

Az elmúlt 50 év legtöbb halálos áldozatát követelő katasztrófatípus a földrengés mellett a hurrikán, és az azt kísérő áradás volt:

3. táblázat. Áradások és hurrikánok [1]

Ország	Dátum	Áldozatok száma
Bangladesh	1970. 11. 12	300000
Bangladesh	1991. 04. 29	138866
Myanmar	2008. 05. 02	138366
Bangladesh	1965. 05. 11	36000
Bangladesh	1963. 05. 28	22000

Bár a táblázatban csak két, fejlődő ország szerepel, a hurrikánok az USA-ban is komoly károkat okoznak és nem kevés emberéletet követelnek, úgyhogy ez is egy olyan katasztrófatípus, aminek a kárait csak enyhíteni lehet, de kiküszöbölni nem.

Hurrikánok és áradások esetén is bekövetkezhet épületomlás, bár ennél a katasztrófatípusnál nem ez a jellemző. Az viszont igen, hogy egyszerre nagy területen hat és megszünteti az áramellátást és a mobilhálózatot is. Ha egy hurrikán alatt valamilyen mentési munkálatot kell végezni, akkor a mentőegységek itt sem számíthatnak a kiépített kommunikációs infrastruktúrákra.

2.1.3. Tűzvész

Két fajta tűzvészt lehet megkülönböztetni:

- Vadtűz (angolul wild fire) esetén egy bozótos, vagy egy erdő gyullad ki. Egy ilyen tűz óriási károkat okoz, de a lakosságot csak ritkán veszélyezteti. Az elmúlt 30 évben világviszonylatban évente átlag 57 ember [2] veszítette életét ilyen jellegű tüzek miatt. Összehasonlításként, Magyarországon 2011-ben 638 ember [3] halt meg valamilyen közlekedési balesetben.
- Az épülettüzek sokkal súlyosabb károkat okoznak a civil lakosságnak is. Az épületben rekedtekkel leggyakrabban a füst és a levegőben terjedő mérgező gázok végznek. 1992-ben Magyarországon minden egymillió emberre 33.1 áldozat jutott. Ezzel magasan vezette a nemzetközi mezőnyt. Magyarország után Finnország statisztikái voltak a legrosszabbak, ahol 21.8 halott jutott egymillió emberre, az USA pedig 19.5 halottal a harmadik volt [4].

Egy épülettűz általában csak néhány lakást, esetleg egy nagyobb épületet érint. Viszonylag könnyű ellene védekezni a korszerű és olcsó automatizált tűzjelző és tűzoltó rendszerekkel, valamint poroltókkal. A probléma az, hogy ezek a berendezések csak irodaházakban és ipari létesítményekben kötelezőek, miközben a tüzesetek háromnegyede lakóházakban következik be.

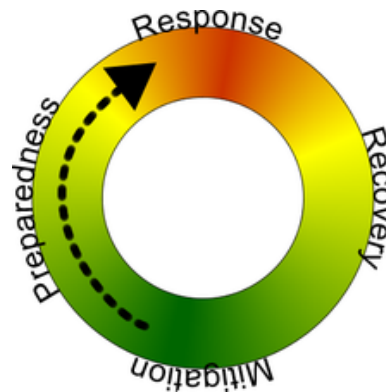
Egy plafonra szerelhető, Bavaria tűzjelző (BARM6) ára 4.000Ft, egy BENTEL H200 tűzjelző sziréna 8.000Ft egy poroltó pedig már kapható 12.000Ft alatt. Érdekes, hogy hiába olcsók ezek az eszközök, mégis meglepően kevés háztartásban találhatóak meg. Ennek

az emberi gondolkodásmód az oka, miszerint hiába jár akár halállal is egy katasztrófaesemény, ha kellően kicsi a (vélt) valószínűsége, akkor az emberek nem szívesen költenek a megelőzésére. (Ezt a gondolkodásmódot az autóipar is szem előtt tartja, ezért alkalmaznak hangjelzést, ha nincs bekötve a biztonsági öv.) Bár a lakástűzek jellemzően nem olyan események, amikkel a dolgozatban megtervezett rendszer foglalkozik, akkor is felhívják egy nagyon fontos gyakorlati követelményre a figyelmet: *Hiába képes a rendszer emberi életek mentésére, ha túl drága, akkor nem fogják használni. A rendszer tervezésekor minél alacsonyabb árra kell törekedni.*

2.2. A túlélők felkutatásának és mentésének módszerei és eszköztára

A katasztrófavédelemnek (ahogy azt a 3. ábra is mutatja) négy fázisa van:

- **Felkészülés:** Ha nincs semmi katasztrófára utaló jel, a hatóságoknak és a civil szervezeteknek akkor is készen kell állni az azonnali cselekvésre.
- **Reakció:** Ha bekövetkezik a katasztrófa, azonnal reagálni kell rá. Ez a védelem legkritikusabb szakasza. A lakosságot evakuálni kell, amennyiben ez nem sikerült, meg kell kezdeni a mentési munkálatokat.
- **Helyreállítás:** Egy katasztrófa után az infrastruktúrában óriási károk keletkeznek. Ezeket minél előbb helyre kell állítani. Ebben a szakaszban fontos megfelelően rangsorolni a prioritásokat. (Például egy kórház áramellátásának helyreállítása sokkal fontosabb, mint egy irodaházé.)
- **Kárenyhítés:** Ebben a szakaszban a hosszabb távú javítási munkálatokat kell befejezni, valamint kezelni kell a civil lakosság katasztrófa utáni tüneteit. Ezek leggyakrabban anyagi természetűek (amiknek a kezelése általában a biztosítók dolga), de előfordulhatnak egészségügyi utóhatások is. Fontos az érintett lakosok hosszú távú megfigyelése.



3. ábra. A katasztrófavédelem fázisai

A dolgozat a legkritikusabb fázisra, a reakcióra (angolul response) koncentrál, hiszen a cél egy olyan rendszer kifejlesztése, ami a krízishelyzet közben nyújt segítséget. Bár rengeteg ilyen rendszer található a piacon (és még több van kutatási fázisban), a katasztrófák hatékony kezelése még nem oldódott meg. A fejezet további néhány kríziskezelő rendszert tárgyal a teljesség igénye nélkül.

2.2.1. Vészkijáratí irányfények

A vészkijáratí irányfények minden nagyobb épületben megtalálhatóak. Látszólag közönséges lámpákról van szó, de valójában egy kifinomult tesztáramkört rejtenek, amik az épületen belül vezetékes hálózatot alkotnak. Erre azért van szükség, mert egy esetleges



4. ábra. Egy vészkijáratí irányfény

épülettűz vagy földrengés esetén az áramellátás megszűnhet. Ha ez éjszaka következik be, vagy a tűz miatt nagy a füst, akkor az épületből lehetetlen kijutni. Ilyenkor az irányfényekben bekapcsol a beépített akkumulátor, amiről egy óráig képesek üzemelni. Egy óra alatt mindenki el tudja érni a vészkijáratot.

A lámpában lévő elektronikára azért van szükség, mert az akkumulátorokat folyamatosan tölteni, kisütni és tesztelni kell. Ha ezeket egy átlagos méretű gyárban (ez 300 lámpát jelent) emberek végeznék, akkor ez hetente két ember teljes munkaidejét igényelné. Hosszútávon olcsóbb (és üzembiztosabb) megoldás egy automatikus rendszert építeni erre a célra.

Ilyen rendszert szállít többek között az olasz Rubic cég. Az ő rendszerük egy központi egységből, a speciális (címezőmodullal felszerelt) lámpatestekből és az őket összekötő sínrendszerből áll. A sín maximális hossza 1200 méter lehet. Ennél nagyobb távolság esetén telepíteni kell egy vagy több jelerősítőt is. Egy darab központi egység képes 1000 darab lámpatest kezelésére. A szükséges tesztek végrehajtása automatizálva van, a felhasználó minden hibáról értesítést kap. A rendszer hátránya, hogy csak vezetékes kommunikációra képes.

Az alábbi táblázatban a rendszer kiépítési költsége látható egy átlagos méretű, fiktív gyár esetén, ahol 300 lámpatestre és 3km sínre van szükség:

4. táblázat. Egy Rubic irányfény rendszer kiépítési költségei

Eszköz megnevezése	Mennyiség	Ár
Rubic központi egység	1db	500.000Ft
Címezőmodulos lámpatest	300db	3.369.000Ft
Jelerősítő	3db	168.450Ft
A lámpák szerelése	2 ember, 6 nap, szerelési anyagok	1.600.000Ft
Kábelezési díj	4 ember, 50 nap, 3km kábel	10.300.000Ft
Kiegészítő szerelvények	Becslés	200.000Ft
Dokumentálás	100 óra	240.000Ft
Élet- és tűzvédelem		300.000Ft

A rendszer teljes költsége majdnem 17.000.000 Forint, de a kiépítésére a nagyobb cégeket, irodaházakat és intézményeket kötelezik. A legtöbb helyen soha sincs rá szükség, de egy

tűz esetén több száz életet menthet meg.

A legtöbb vészhelyzeti rendszernek éppen itt mutatkozik meg a gyengesége: nagyon drága a telepítés és valószínűleg soha sem lesz rá szükség. A cégek és intézmények egyszerűen nem látják értelmét a kiépítésének. Ezért fontos, hogy egy ilyen rendszer minél olcsóbb és egyszerűbb legyen, valamint az is egy nagy előny, ha több (nem feltétlenül a katasztrófavédelemhez kapcsolódó) funkciót is ellát.

2.2.2. Tűzjelző eszközök

Az előző fejezetben már volt szó a tűzvédelem fontosságáról. Ebben a pontban egy konkrét megoldásról, a Siemens Cerberus Pro rendszerről lesz szó.



5. ábra. Cerberus Pro

A termékcsalád nagy előnye a jó skálázhatóság. A vevő kiválaszthatja a számára ideális központot, beállíthatja a tűzjelzők hálózati topológiáját és később bővítheti a rendszert. Kompatibilis szenzorok terén is nagy a választék. Lehetőség van nagy érzékenységgű, több szenzorral felszerelt vagy egyszerűen csak olcsó eszközök beszerzésére is. A Cerberus Pro célcsoportja az irodák, nagyobb oktatási intézmények és ipari létesítmények.

A rendszer hátránya, hogy a modulok csak vezetékesen tudnak kommunikálni a központtal. Ez egyrészt azt jelenti, hogy egy régi épület lefedése nagyon költséges lehet, másrészt pedig azt, hogy a vezetékek sérülése esetén (amire egy épülettűzben nagy esély van) a rendszer nem képes pontosan jelezni a tűzfészkek helyét, hiszen szétesik a hálózat. Ez utóbbi önmagában nem nagy hátrány, hiszen ezeket a rendszereket általában csak a tűz kezdetekor szokták használni, amikor még a falak érintetlenek.

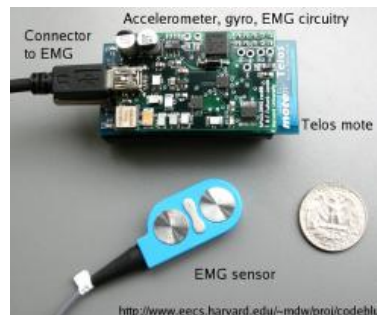
Nagyobb a gond, ha a tűz más katasztrófák következménye. Földrengések vagy merényletek gyakran okoznak épülettűzet is. Ilyen esetekben egy vezetékes rendszer hatástalan lehet. Ezért fontos, hogy egy általános katasztrófavédelmi rendszer (a baj bekövetkezte után) ne támaszkodjon semmilyen meglévő infrastruktúrára. Jó példa erre az előző pontban tárgyalt irányfény-lámpa, ami normális esetben a hálózati feszültségről tölti az akkumulátorait és egy vezetékes hálózaton jelzi a hibákat, baj esetén viszont mindentől függetlenül képes működni.

2.2.3. CodeBlue

A dolgozat eddigi példái kereskedelmi forgalomban kapható, kipróbált eszközökről szóltak. Ebben a pontban viszont Harvard WSN laborjának az egyik fejlesztése, a CodeBlue kerül bemutatásra.

A CodeBlue [5] projekt célja, hogy bemutassa, hogyan lehet felhasználni a WSN-rendszereket

a mindennapi gyógyításban. Az alapötlet az, hogy az emberek viseljenek egy kis méretű szenzort, mely folyamatosan vizsgálja az életjeleiket. A modul tartalmaz egy saját fejlesztésű, kis fogyasztású pulzuszámológát és egy EKG-t. Ezek a szenzorok folyamatosan gyűjtik az adatokat és broadcast-üzenetként elküldik minden olyan adatkoncentrátornak, ami képes fogadni a jeleket. (Az RF-modulok hatótávolsága kb. 100 méter.) Az adatkoncentrátorok lehetnek PDA-k, laptopok vagy akár sürgősségi-terminálok is.



6. ábra. CodeBlue szenzor

Az adatkoncentrátorok képesek az adatokat valós időben, grafikonok formájában megmutatni és eltárolni. (Ezzel később segítheti az orvos munkáját.) A szenzorok nem csak mérésekre képesek, hanem alapvető feldolgozást is végre tudnak hajtani: minden szenzorhoz be lehet állítani egy normális tartományt. Ha a mért értékek nem esnek bele ebbe a tartományba, a modul azonnal riaszt.

A CodeBlue elsődleges célja, hogy segítse az orvosokat betegek megfigyelésében, de nem használ ki minden lehetőséget, amit egy WSN-rendszer biztosít. A CodeBlue egy lehetséges továbbgondolása az Aid-N projekt, amit katasztrófa-helyzetek kezelésére fejlesztettek ki.

2.2.4. Aid-N projekt

Az Advanced Health and Disaster Aid Network [6] (vagy röviden Aid-N) projekt célja, hogy egy esetleges katasztrófa bekövetkezte után a mentőcsapatokat segítse az áldozatok felkutatásában és az egyes esetek osztályozásában. Ahogy arról már volt szó, az Aid-N a CodeBlue továbbgondolása. A szenzorok felépítése a két rendszerben nagyon hasonló, de az Aid-N kibővíti a lehetőségeket. Képes például a térerősségen alapuló helymeghatározásra is, ezzel segítve a romkutatók munkáját, akik akár a romok alatt rekedt áldozatok életjeleit is figyelhetik. A rendszer fő problémája az, hogy mindenkinek viselnie kell ezt a szenzort, bárhová is megy. Ez két aggályt vet fel:

- 1.) a szenzor (az elemek miatt) nem kicsi. A viselése kényelmetlen lehet.
- 2.) a csomópontok elsődleges feladata az, hogy a viselőjének a pozícióját meghatározzák, ami lehetőséget biztosít különböző visszaélésekre. Az emberek nagy része jobban fél a folyamatos és biztos megfigyeléstől, mint egy nagyon kis valószínűségű épületomlástól.

2.2.5. Rescue cell

Ma már szinte minden embernél van mobiltelefon, aminek a helyét viszonylag könnyű a cellainformációk alapján bemérni, amit akár a mentőegységek is kihasználhatnak a

túlélők keresése közben. A módszer problémája az, hogy a romok (vagy lavina esetén hótorlasz) alatt rekedt telefon képtelen átküldeni a jelet az akadályon, így nem állnak rendelkezésre cellainformációk. (Ráadásul az sem biztos, hogy van a közelben működő bázisállomás.) Ezt a problémát képes áthidalni a Rescue cell [24] megoldása azzal, hogy a mentőegységeket felszereli egy kicsiny, csuklóra szerelhető bázisállomással. Ezek a bázisállomások elég közel kerülnek a romok alatt rekedt telefonokhoz, így képesek felvenni velük a kapcsolatot. Ha legalább négy mobil bázisállomás kapcsolatba lépett ugyan azzal a telefonnal, akkor hozzávetőlegesen meg lehet határozni a helyzetét.

A Rescue cell rendszer jelenleg egy aktívan kutatott és támogatott kísérleti megoldás, mely az Európai Unió finanszírozásában, nemzetközi együttműködéssel készül. Magyarországon a Rubin Informatikai Zrt. a központi egység fejlesztésében vesz részt, melynek feladata, hogy a mentőegységek adatait rendszerezze és a lehetséges túlélők helyzetét megjelenítse egy háromdimenziós térképen.

2.2.6. Légzésdetekció

Ha valaki élve a romok alá szorul, valószínűleg nem tud mozogni, gyakran az eszméletét is elveszíti, viszont lélegezni biztosan fog, ezért logikus, ha a keresőegységek a légzés detektálására koncentrálnak. (Ha az áldozat légzése is leáll, akkor a mentőcsapatoknak lehetetlen időben elérniük ahhoz, hogy megmenthessék.) A légzés detektálására több különböző módszer létezik és jelenleg is fejlesztenek ilyen célú eszközöket.

- 1.) Egyrészt ha valaki lélegzik, akkor széndioxidot bocsát ki. Ez a gáz éghetetlen, kémiaiilag stabil, tehát megvan az esélye, hogy kijut a romok közül, ahol egy megfelelő szenzor detektálhatja. Természetesen vannak egyéb anyagok is, amik csak egy élő személytől származhatnak (izzadság, ammónia), így elvileg bármelyiket figyelve megtalálhatóak a romok alatt rekedt emberek. A mentőkutyákat ezeknek a szagoknak az érzékelésére képezik ki, ez azonban nagyon költséges folyamat nem beszélve arról, hogy a kutya és a gazdája komoly veszélyben van a romok között. Az angliai Loughborough Egyetemen egy, a kutyákat kiváltó szenzor kifejlesztése folyik, de egyelőre még csak részeredményeket értek el [7].
- 2.) Valamivel egyszerűbb, ha a sérült mozgását próbálják figyelni. Ez történhet például radarszerű eszközökkel is [8]. A módszer során a mentőegységek egy speciálisan kialakított radarral figyelik a romokat és különböző változásokat keresnek (7. ábra, bal oldal). Ha valahol megváltozik a radarhullámok visszaverődése, akkor ott valószínűleg valaki mozog (7. ábra, jobb oldal). Ez a módszer már bizonyított. Az 1995-ös japán földrengésben ilyen speciális radarokkal is kutattak a túlélők után. A rendszer hátránya, hogy nagyon rövid a hatótávolsága, így a felszíntől távol lévő túlélőket nehéz vele felkutatni.



7. ábra. A bal oldali ábrán egy romkutató keresi a romok alatt rekedteket, a jobb oldali ábrán pedig az általa használt radar kimenete látszik. Az ábra felső részén jól látszik, ahol változik a visszavert hullám intenzitása, ez valamilyen mozgásra utal.

- 3.) Egy harmadik variáció a túlélők keresésére a romok alá juttatott mikrofon. Egy kellően érzékeny mikrofon akár méterekről is képes érzékelni a túlélők keltette rezgéseket. Az így felvett hangok alapján egy képzett személy képes eldönteni, hogy van-e valaki a romok alatt, vagy sem. Ez a módszer hasonlít a rádióhullámos detekcióra abban, hogy itt is hullámformákat kell feldolgozni, de ebben az esetben a jelfeldolgozás sokkal nehezebb. Ráadásul ez is rövid hatótávolságú, ugyanakkor előnye, hogy a mikrofon kis méretű, ezért könnyen bejuttatható a romok alá.

Ahogy az a felsoroltakból látszik, a katasztrófhelyzetek kezelése egy nagyon aktuális kutatási terület, amiben egyelőre nem született olyan eredmény, ami minden felmerülő kérdést megoldana. Szükség van új rendszerekre és módszerekre, amik megkönnyítik a túlélők detekcióját és segíti a mentési munkálatokat. A dolgozat célja egy olyan rendszer megtervezése, ami automatikus mérésekkel segíti a túlélők hang alapú detektálását, a mérési eredményeket pedig WSN-nel juttatja el a mentőegységekhez.

Ehhez egyrészt ki kell fejleszteni egy, a túlélők detektálására képes módszert, másrészt egy ezt támogató hálózati topológiát. A továbbiakban ezekről lesz szó.

3. Vezeték nélküli érzékelőhálózatok

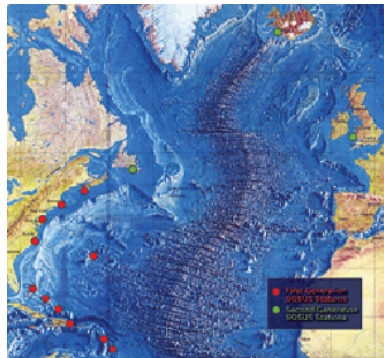
Az információs tudományok az elmúlt évtizedekben robbanásszerű fejlődésen mentek keresztül. Ma már kereskedelmi forgalomban kaphatóak olyan pénztárca méretű OBC-k (OnBoard Computer), melyek kapacitása nagyobb egy kilencvenes évekből ránk maradt PC-nél [10]. Ez a technológiai fejlődés olyan alkalmazási lehetőségeket ad a mai mérnökök kezébe, amikre 20 évvel ezelőtt még nem is gondolhattak. Egy ilyen alkalmazás a civil felhasználású vezeték nélküli szenzorhálózat is (továbbiakban WSN).

3.1. Történeti bevezető

A szenzorhálózatok ötlete persze nem új. Az embereknek mindig szükségük volt arra, hogy a környezetükről friss és pontos információkkal rendelkezzenek. Már az ókori hadviselés során is csaták kimenetelét dönthette el egy-egy jól kiépített információs útvonal, mellyel a parancsnok tájékozódhatott a sereg helyzetéről és szükség esetén utasításokat is küldhetett a frontvonal bármely részére. Bár szerepe szerint ez az emberekből, dobokból és kürtökből álló hálózat is tekinthető szenzorhálózatnak, a mai értelemben vett WSN valódi őseinek az 1950-es évek elején megalkotott SOSUS-t [11] tartják.

3.1.1. SOSUS

A SOSUS (Sound Surveillance System) egy akusztikus érzékelőhálózat, melynek az érzékelőit az Atlanti- és a Csendes-óceánban helyezték el. A rendszer feladata az volt, hogy folyamatosan figyelje a tengeralattjáró-tevékenységet.



8. ábra. Az első SOSUS állomások [12]



9. ábra. Egy SOSUS-mérőállomás [12] (1957)

A 8. ábra alapján jól látható a hálózat alapkoncepciója: az egyes állomások ultrahangnyalábokkal pásztázzák az óceánt. Ha egy állomás befog egy ismeretlen objektumot, akkor csak a relatív távolságát tudja mérni. Tizenegy darab hálózatba kötött állomás viszont nagy pontossággal képes meghatározni egy esetleges behatoló helyzetét és mozgását.

A SOSUS egy modern és előremutató rendszer volt, azonban a kor technológiai korlátjai miatt (a 9. ábrán látható épület az egyik mérőpont) ez a civil szférában nem terjedhetett el.

3.1.2. ADSID

A szenzorhálózatok következő mérföldkövének nevezhető eszköz az ADSID (Air DeAir Delivered Seismic Intrusion Detector) nevű rendszer volt, melyet szintén a katonaság fejlesztett ki [11]. A US Air Force használta a vietnámi háborúban (1975) az ellenséges csapatmozgások felderítéséhez.

Ebben már felismerhető néhány, a modern WSN-rendszerekre is jellemző megoldás. A rendszer több vezeték nélküli geoszenzorból és egy adatfeldolgozó egységből állt. Mind-egyik szenzornak volt egy saját frekvenciasávja (FDMA), amin keresztül el tudták küldeni a szeizmológiai adatokat egy közeli reptérre. (Csillag topológia.) Egy szenzor 1.2 méter hosszú, 0.2 méter átmérőjű és 17 kg tömegű volt. Ilyen méretek és körülmények mellett fel sem merültek a WSN-re jellemző mai problémák, úgy mint a spektrum minél effektívebb kihasználása, vagy a tápforrás kímélése.

3.1.3. Újgenerációs WSN

A mai értelemben vett WSN nem jöhetett volna létre a nagy integráltságú CMOS-technológia nélkül [11]. Az első olyan termék, ami ezt ki is használta, az UCLA és a Rockwell Science Center által piacra bocsátott LWIM (Low Power Wireless Integrated Microsensors) termékcsalád volt. Ennek a terméknek a legnagyobb innovációja az általánosság: ahelyett, hogy célhardvereket építettek volna egybe, egy általános mikrovezérlőt és egy RF modult integráltak egyetlen lapkára.

Az LWIM csomópontjai 10m-es hatótávolsággal, 100kbps-es adatsebességgel és 1mW-os adóteljesítménnyel működtek. Bár ezek az értékek mai szemmel nézve nem túl jók (összehasonlításként a Digi XB24 moduljai szintén 1mW fogyasztás mellett akár 300 méterre is képesek elküldeni az adatokat 500 Kbps-sel [13]), ez a termékcsalád indította el a mai értelemben vett WSN térhódítását.

3.2. Protokollok

Egy WSN rendszer működési környezetéről a gyártás során nem rendelkeznek semmilyen információval. Egy ilyen rendszernek önálló döntéseket kell hoznia, melynek során megtervezi az információs útvonalakat, az egyes csomópontok feladatait, gondoskodik a kieső csomópontok pótlásáról és átszervezi magát, ha szükséges. A döntések meghozatalát és a kommunikáció pontos részleteit a protokollok írják le. Ebben a fejezetben a protokollokról, mint általános fogalomról lesz szó.

3.2.1. A protokolltervezés kihívásai WSN esetén

Egy protokoll megtervezése folyamán pontosan meg kell határozni a megoldandó problémát és a megszorításokat. Egy WSN esetén ezek elsősorban a következők [19]:

- **Adatközpontú paradigma:** Az eddig megszokott és elterjedt hálózatok (LAN, MAN, WAN, ...) elsődleges feladata az, hogy a kommunikáció két résztvevője között megtalálják a legrövidebb utat. A WSN-ben viszont nem két résztvevő, hanem a források és (általában) egy nyelő közötti adattovábbítást kell megoldani.
- **Együttműködő útvonaltervezés és adatfeldolgozás:** Az információ útvonalának megtervezése és feldolgozása egymástól nem elválasztható funkciók. Az infor-

máció tartalmától függően kell meghatározni az útvonalát.

- **Energiahatékony tervezés:** A WSN csomópontok akkumulátorait telepítés után nagyon költséges, vagy lehetetlen újra feltölteni. Éppen ezért az energia gazdaságos felhasználása elengedhetetlen a hálózat hosszútávú működéséhez.
- **Hálózatfelderítés és -szervezés:** Egy WSN sokkal nagyobb egyetlen node hatókörénél. Az egyes csomópontok csak a közvetlen szomszédaikat látják. Ahhoz, hogy az információ eljusson a forrásoktól a nyelőig, meg kell tervezni az útvonalat. Az útvonalak megtervezéséhez az összes csomópont együttműködése szükséges. Ráadásul az útvonal tervezését nem elég egyszer elvégezni, hiszen a hálózat topológiája dinamikusan változhat.
- **Biztonság:** Mivel a WSN csomópontjai általában könnyen hozzáférhetőek, ráadásul nyílt szabványokat használnak, alkalmazástól függően gondoskodni kell a megfelelő titkosításról. (A dolgozat szempontjából az adatbiztonság nem releváns kérdés.)

3.2.2. OSI modell

Az OSI modell egy rétegzett modell, ami absztrakt leírása egy hálózati protokollnak. Bár WSN esetén nem mindig ad optimális megoldást [20], egyszerűsége és modularitása miatt előszeretettel használják egy protokoll tervezése folyamán.

Az OSI modell hét réteget különböztet meg, amiből általában ötöt használnak. Az egyes rétegek jól definiált interfészek segítségével tudnak kapcsolatba lépni a közvetlenül alattuk és felettük elhelyezkedő rétegekkel. Ezek neve és rövid leírása:

- 1.) **Fizikai réteg** : A kommunikációs közeg specifikálja a jelek terjedésének közegét. Ez lehet levegő, víz, vákuum, vezeték, stb. . . A kommunikációs technológia specifikálja, hogy a digitális bitek milyen formában kerüljenek rá az analóg csatornára. A réteg különböző szempontok alapján tud optimalizálni. Ilyen szempont lehet az átviteli energia, az átviteli sebesség, stb. . .
- 2.) Az **adatkapcsolati rétegen** belül a Logical Link Control gondoskodik a hibajavításról, valamint kapcsolatot tart a felette és alatta lévő réteggel. (Kezeli a frame-eket.) Vezeték nélküli rendszerekben a MAC-réteg felelős a címzésért, az adatok célba juttatásáért, valamint vezérelheti az átviteli energia szintjét.
- 3.) A **hálózati réteg** felelős a csomagok célba juttatásáért és az útvonalak kiválasztásáért. Legfőbb nehézsége, hogy egy megbízhatatlan vezeték nélküli kapcsolat felett kell megbízható kommunikációt biztosítani.
- 4.) A **transzport réteg** a hálózati réteg és az alkalmazás réteg között biztosít egy absztrakt interface-t, amin keresztül az alkalmazásréteg képes csomagokat küldeni és fogadni anélkül, hogy foglalkoznia kellene a konkrét közeggel.
- 5.) Az **alkalmazás réteg** szolgáltatásokat biztosít a végfelhasználó alkalmazása felé és ezzel biztosítja a hatékony kommunikációt a végpontok között.

4. Gyakorlatban használt protokollok

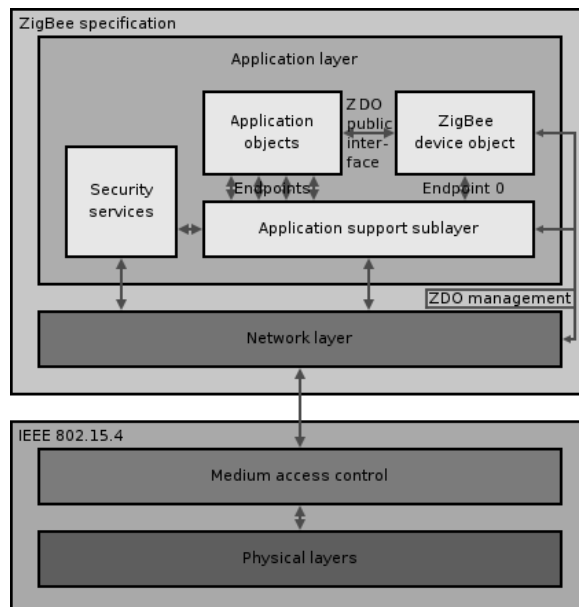
Napjainkra a különböző WSN-rendszerek a mindennapjaink szerves részét képezik. Ebben a fejezetben két elterjedt protokoll kerül bemutatásra, melyekre a különböző WSN-rendszerek előszeretettel építenek.

4.1. ZigBee

A ZigBee kifejlesztése során egy olyan nyílt protokoll kidolgozása volt a cél, mely az alábbi követelményeknek felel meg [21]:

- nagy hálózatok működtetése, melyek akár évekig is működnek operátori beavatkozás nélkül;
- alacsony energiafogyasztás, alacsony eszköz- és beüzemelési költség, alacsony komplexitás és kis méret;
- a konkurens termékekhez és szabványokhoz képest alacsony adatátvitel;
- szabványosított protokoll, melyet több gyártó is felhasználhat;

A protokoll vázlatos rajza a 10. ábrán látható. A továbbiakban először az egyes rétegekről [21], majd néhány konkrét alkalmazásról lesz szó.



10. ábra. ZigBee protokoll.

4.1.1. Fizikai réteg és MAC réteg

A fizikai és a MAC réteget az IEEE 802.15.4-es szabvány adja meg. Ez a szabvány kellően robusztus egy szenzorhálózathoz. Az adatok kódolásához a fizikai rétegben O-QPSK-t és DSSS-t (Offset-Quadrature Phase-Shift Keying és Direct Sequence Spread Spectrum) használ. A két technológia kombinációja kitűnő teljesítményt nyújt alacsony SNR (Signal noise ratio) esetén is.

A MAC-réteg az ütközések elkerülésére CSMA-CA-t használ (Carrier Sense Multiple Access Collision Avoidance), melynek során a modul küldés előtt belehallgat a csatornába és addig nem kezdi meg a saját üzenetének az adását, amíg hall valamit.

4.1.2. Hálózati réteg

A hálózati réteget már a ZigBee specifikáció adja meg.

A csomagok hibamentességét egy 16 bites CRC-kód garantálja. Ha a vett csomag CRC-kódja hibás, akkor a fogadó fél tájékoztatja a hibáról a forrást, aki megismételheti a küldést.

Egy csomagot négyszer próbál meg továbbküldeni egy modul. Ha negyedszerre sem sikerült, az eredeti forrás kap egy visszaigazolást a sikertelen küldésről. (Ez sikeres küldés esetén is megtörténik, egyrészt minden hop után, másrészt azután, hogy a csomag elérte a tényleges célját.)

A megbízhatóságot növeli az automatikusan felépülő mesh-hálózat is, melyben a ZigBee operátori segítség nélkül megtalálja a legjobb útvonalat. Ha az útvonal egyik tagja a későbbiekben megsérül, a hálózat keres egy újat.

4.1.3. Alkalmazásréteg

Egy ZigBee hálózatban háromféle csomópont van, melyek feladatát az alkalmazásréteg határozza meg:

- 1.) **Koordinátor (ZC):** A hálózat felépítéséért és vezérléséért felelős, általában ez a nyelző. A mérések ide futnak be. Kitüntetett címe (0x0) van a hálózatban.
- 2.) **Router (ZR):** A hálózat határainak kibővítéséért felelős, képes más eszközöket fogadni.
- 3.) **End device (ZE):** Általában ez végzi a méréseket. Ez az egyetlen tagja a hálózatnak, aminek engedélyezett az alvás. Nem képes további eszközöket kiszolgálni. Azért nagy a jelentősége, mert ez az a csomópont, ami akár évekig is képes üzemelni egy pár AA-elemmel.

Összefoglalva, a ZigBee egy jól átgondolt, bizonyított, nyílt szabvány, mely általában jó választás egy WSN számára.

4.1.4. Alkalmazások

A ZigBee az elmúlt években széles körben elterjedt az iparban és a civil szférába is kezd betörni. Ez a fejezet néhány ipari alkalmazást mutat be, a teljesség igénye nélkül.

- **Libelium: Erdőtűzfigyelő rendszer [14]** A Libelium egy automatizálással foglalkozó cég, ami elsősorban Digi ZigBee alapú érzékelőhálózatokat fejleszt és értékesít [15]. Ez a cég fejlesztett ki 2009 júliusában egy olyan szenzorhálózatot, mellyel erdőkben képes detektálni a tüzet még azelőtt, hogy az komolyabban elterjedne. Az erdőtüzeket hagyományosan kamerarendszerekkel figyelik. Ezek nagyon drága műszerek, ráadásul csak akkor veszik észre a tüzet, amikor már elérte a fák lombkoronáját [16].

A Libelium által kifejlesztett szenzorrendszer ezzel szemben a tűz hatásaira figyel, úgy mint a páratartalom és a különböző gázkoncentrációk megváltozása. A fákon elhelyezett szenzorok által mért adatok egy, a tűzoltóságnál lévő szerverhez futnak be, ami fogadja és értelmezi őket. Ha valamelyik gáz koncentrációja hirtelen megemelkedik, akkor riasztást küld a Parancsnoki Központba. A vezeték nélküli hálózat szükséges választás volt, hiszen egy egész erdőt bekábelezni nem lehetséges.

- **Libelium: Közlekedésmonitorozás Bluetooth szenzorokkal [17]** A Libelium egy másik érdekes terméke ötvözi a ZigBee és a Bluetooth képességeit ahhoz, hogy képes legyen figyelni a forgalmat egy átlagos, nyugat-európai városban. A csomópontok egy ZigBee hálózaton keresztül juttatják el az áthaladó forgalomra vonatkozó méréseiket, az autókat pedig a ZigBee mellett elhelyezett Bluetooth modul érzékeli. A mérés arra a feltételezésre alapul, hogy egy átlagos, nyugat-európai autóban van legalább egy bekapcsolt bluetooth-képes készülék. Természetesen ez nem ad olyan pontos eredményt, mint egy úttest alá épített indukciós tekercs, cserébe a bekerülési költsége az előbbihez képest elenyésző. A vezeték nélküli hálózat egy jó választás volt a probléma megoldásakor, hiszen így egy új mérési pont felállításához nem kell új kábeleket lefektetni.
- **Gamma Digital: GD Sentinel Box [18]** A Gamma Digital egy magyar cég, mely egyedi műszergyártással, irányítástechnikával és automatizálással foglalkozik. A GD Sentinel Box egy általuk kifejlesztett, szintén ZigBee alapú, beltéri minőségbiztosító rendszer. A rendszer vezeték nélküli csomópontjai szabályos időközönként hőmérséklet és páratartalom méréseket végeznek, a mérési eredményeket pedig egy szerver elraktározza addig, amíg a felhasználó erre igényt tart.

A rendszer célsoportja a különböző élelmiszer- és gyógyszeripari cégek, akiknek fontos biztosítani és bizonyítani a termékeik megfelelő raktározását. A raktárhelyiségekben elhelyezett, hitelesített mérőpontok elméletben használhatnának vezetékes átvitelt is, ehhez azonban ki kellene építeni az ehhez szükséges infrastruktúrát, ami nem költséghatékony.

4.1.5. A ZigBee korlátai

A ZigBee tökéletes arra, amire tervezték: egy szenzorhálózatban a mérőpontokból (ZE) eljuttatja az adatot a nyelőbe (ZC), ugyanakkor ha két csomópontnak kell egymással kommunikálnia, akkor az nem hatékony. Azonkívül a folyamatosan változó hálózati topológiához sem képes elég gyorsan alkalmazkodni. Mivel a legtöbb WSN-ben adatáramlás csak a nyelő felé történik és a csomópontok helyzete rögzített, ez nem jelent problémát.

4.2. Bluetooth

A Bluetooth eredetileg mobil és fix eszközök nagy sebességű és fokozott biztonságú összeköttetésére jött létre [22]. Több eszközt is képes összekötni egy hálózatba. Akár a ZigBee, mára már ez is egy cégektől független, nyílt szabvány lett. Rengetegen gyártanak Bluetooth-kompatibilis chipeket.

A Bluetooth protokollt nem a WSN igényeinek megfelelően fejlesztették ki: nem tesz lehetővé tetszőlegesen nagy hálózatokat (egy piconetben egy mester és 7 szolga lehet egyszerre jelen) és csak a csillagtopológiát támogatja. Ettől függetlenül, napjaink egyik meghatározó vezeték nélküli technológiája, ezért fontos bemutatni az itt alkalmazott megoldásokat, bár nem olyan részletesen, mint a ZigBee esetében.

4.2.1. Fizikai réteg

A Bluetooth szórt spektrumot használ, ezzel ellenállóbb a belső helyiségekben tapasztalható fading-re. A 2.0-ás verzió előtt csak a GFSK (Gaussian frequency-shift keying) modulációt támogatta, amivel képes volt elérni az 1Mbit/s adatátviteli sebességet.

4.2.2. MAC réteg

A Bluetooth egy csomagalapú protokoll mester-szolga struktúrával [23]. A mester (minden pikohálózatban egy) egy szinkronizációs órajelet küld minden $312.5\mu s$ -ban. Két órajelet meghatároz egy $625\mu s$ -nyi időrést. Ekkora időrésekre kell felosztani egy csomagokat. Egy réspár hossza $1250\mu s$. Az réspár első szeletében a mester küld, a másodikban pedig valamelyik szolga. Egy csomag 1, 3 vagy 5 résnyi hosszú lehet.

A mester-szolga szerep (ellenben a ZigBee koordinátor, router, end-device leosztásával) nem kötött, a kommunikáció során felcserélődhetnek.

4.2.3. Scatternet - Piconet

A Bluetooth protokollja lehetőséget biztosít úgynevezett scatternet létrehozására is, ami több piconet összekapcsolásával jön létre. Ilyenkor bizonyos eszközök az egyik piconetben mestert, a másikon pedig szolgát játszanak. Csak kevés alkalmazás használja ki ezt a funkciót.

4.2.4. Alkalmazások

A Bluetooth protokollt széleskörűen alkalmazzák olyan helyzetekben, amikor néhány, fizikailag is közel lévő eszközt kell összekapcsolni és közöttük egy nagy sebességű, ideiglenes hálózatot létrehozni.

Különböző Bluetooth-csipek megtalálhatók az újonnan gyártott mobiltelefonokban, headsetekben, autókban, laptopokban és nagyon sok egyéb hordozható eszközben.

4.2.5. A Bluetooth korlátai

A Bluetooth tökéletesen működik olyankor, amikor kisszámú, fizikailag egymáshoz közeli eszközt kell ideiglenesen összekapcsolni. Klasszikus WSN-rendszerek esetén nem érdemes használni, mert rövid a hatótávolságuk, korlátozott a hálózatok mérete és nagy a protokoll energiaigénye. Ezekről függetlenül PAN esetén a Bluetooth a legelterjedtebb szabvány.

5. Hang alapú túlélőkeresés vezeték nélküli érzékelőhálózatokkal

A második fejezetben bemutatott helyzetekből és megoldásokból látszik, hogy egy katasztrófa után a túlélők megkeresése egy fontos, de nehéz feladat. A mentőcsapatok szenzorai nem képesek elég mélyre hatolni, nem elég érzékenyek és nehéz egy olyan, könnyen mérhető fizikai paramétert találni, ami egyértelműen utal a túlélők jelenlétére. Ráadásul a mérések elvégzése általában emberi beavatkozást igényel, ezért a folyamat egyrészt lassú, másrészt a kutatócsapat állandó veszélyben van.

A továbbiakban egy olyan, WSN-alapú rendszerről lesz szó, ami a fenti problémákat igyekszik orvosolni:

- A szenzorokat még a katasztrófa bekövetkezte előtt, az épületben szét kell szórni, így azok a mérés szempontjából ideális helyen lesznek.
- A rendszer emberi beavatkozás nélkül képes megkeresni a túlélőket.
- A szenzorok megfelelő hangmintákat keresnek, amik csak túlélőkre utalhatnak.

Ezekhez az kell, hogy a szenzorrendszer nagyon olcsó legyen, megfelelő legyen a számítási kapacitása egy komplex jelfeldolgozó algoritmus futtatásához és képes legyen a mért információt eljuttatni a mentőegységekhez.

A dolgozatban megtervezett rendszer a hatékonyabb megvalósítás érdekében nem az emberek által kiadott hangokat, hanem egy mobiltelefon által generált hangmintát fog detektálni. Ez nem okoz problémát, mert a mobiltelefon nagyon elterjedt. 2011-ben majdnem hatmilliárd embernek volt mobiltelefonja (az előfizetők száma alapján), amit szinte mindenhová magukkal visznek. Így a mobiltelefon keresésével megtalálható annak tulajdonosa is [24].

A fejezet további részeiben először a hangok detektálásáról, majd a túlélők beméréséről, végül a rendszer architektúrájáról lesz szó.

5.1. Hangdetekció és lokalizáció

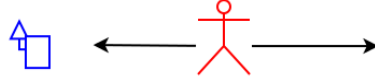
Ahhoz, hogy mikrofonokkal embereket lehessen keresni egy területen, három dolognak kell teljesülnie:

- 1.) a mikrofonok helyét ismerni kell;
- 2.) a mikrofonoknak hallaniuk kell a mobilok által kiadott hangokat;
- 3.) a mikrofonoknak elég sűrűn kell lenniük a helymeghatározáshoz.

Ennek az alpontnak a célja, hogy bemutassa a hang alapú bemérés módszerét, ezért az előző feltételek meglétét nem vizsgálja. Egyelőre csak jól meghatározott frekvenciájú hang (sípólás) beméréséről lesz szó, de a módszer könnyen átalakítható tetszőleges hangmintára. (A mikrofonok helymeghatározásáról egyáltalán nem lesz szó, de léteznek módszerek [25] arra, hogy egy WSN-en belül a csomópontok meghatározzák a helyzetüket.)

5.1.1. Egy mikrofon, egy hangforrás

Először érdemes a lehető legegyszerűbb elrendezést megvizsgálni, melyben egy darab mikrofon és hangforrás található. Utóbbinak ismert a hangerőssége és a frekvenciája. (Lásd 11. ábra.)

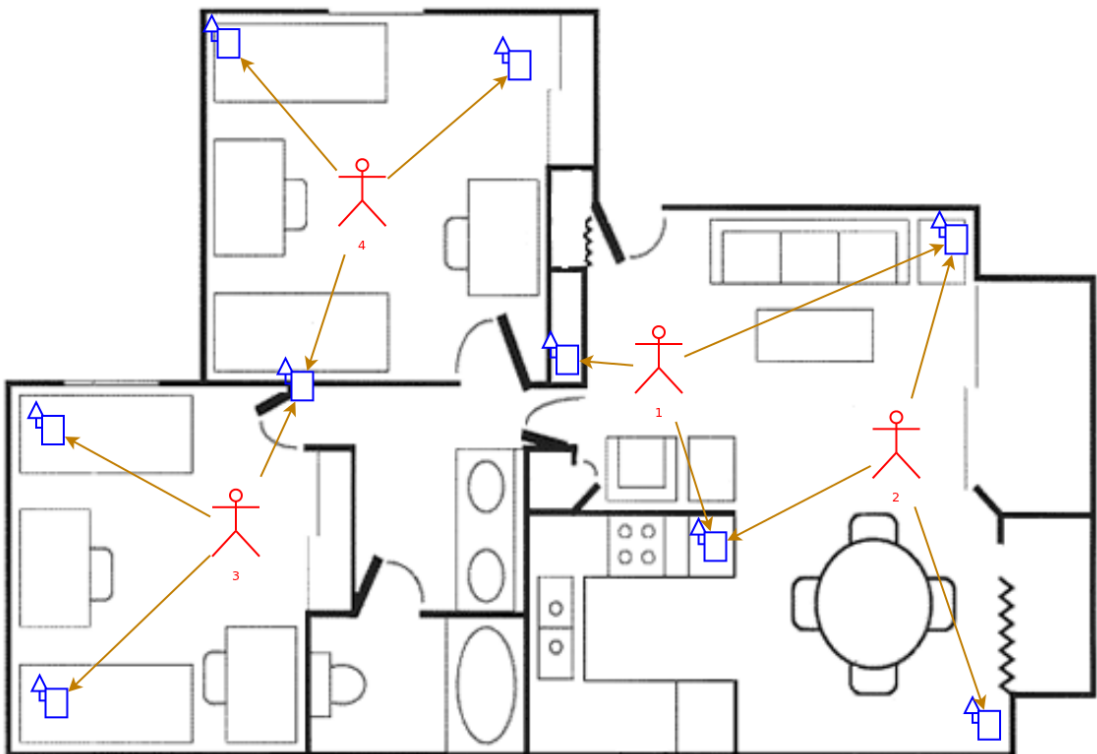


11. ábra. Egy mikrofon és egy hangforrás. A közöttük lévő távolság változik

Ebben az elrendezésben nincs interferencia. (Hangok detektálásánál is a vezeték nélküli kommunikációban megismert problémákkal kell számolni, hiszen itt is hullámterjedésről van szó.) Néhány egyszerű méréssel be lehet bizonyítani, hogy a hangforrás távolsága (néhány méterig) jól becsülhető a mikrofon által mért hangerősséggel. Ezen mérések pontos leírása és eredménye megtalálható a 7.1. fejezetben.

5.1.2. Több mikrofon, több hangforrás

Adott a 12. ábrán látható elrendezés, melyben négy ember és kilenc szenzor (mikrofon) látható.



12. ábra. Beszenzorozott (kék) épület négy emberrel (piros)

Az ábrán látszik, hogy az emberek a mikrofonoktól különböző távolságokra vannak és van olyan mikrofon, ami több ember hangját is hallja. Az ábrán narancssárga nyíl jelöli a hang egyenes terjedését, de nem jelöli a fading-et és az interferenciákat. Minden mikrofon

a hozzá elérő hangok összegét hallja. Ez az az elrendezés, ami a legjobban hasonlít egy valós helyzetre. Ezen belül két szituációt lehet megkülönböztetni:

- 1.) Több a hangforrás, mint a mikrofon: Ekkor minden csomópontnak több ember jelét is fel kellene dolgoznia. Ilyenkor az interferenciával számolni kell, ami nagyon megnehezíti a helymeghatározást.
- 2.) Több a mikrofon, mint a hangforrás: Ez esetben elég, ha a mikrofonok érzékelési sugara csak néhány méter. Ekkor nem nagyon lesz olyan mikrofon, ami egyszerre 3-4 ember jelét is fogja, ezért az interferenciával nem kell foglalkozni.

Figyelembe véve, hogy a rendszernek romok alatt kell működnie, ahol a hangok terjedése nagyon rossz, a szenzorokat mindenképpen sűrűn kell telepíteni, négy-öt méterre egymástól, ezért a gyakorlatban csak a második eset fordulhat elő, ami könnyen visszavezethető a 11. ábrán látható elrendezésre.

Gondolni kell arra is, hogy gyakran van egymás mellett több ember. Ez esetben hiába van sok szenzor az épületben, előfordulhat, hogy az egymáshoz közeli telefonok jele zavarja vagy kioltja egymást. Erre a problémára egy lehetséges megoldás, ha a mobiltelefonok nem adnak folyamatos hangjelzést. Ennek a pontos megvalósításáról a 8. fejezetben lesz szó.

5.2. Hangfeldolgozás

Az előző fejezet bemutatta, hogy hogyan lehet egy sűrűn telepített mikrofonrendszerrel meghatározni egy hangforrás pozícióját, ebben a fejezetben pedig arról lesz szó, hogy hogyan lehet optimálisan feldolgozni a hangot.

Ebben a fejezetben csak a hangfeldolgozó algoritmusokról és a számukra szükséges minimális hangminőségről lesz szó, az algoritmusok implementálásáról nem. Mivel egy WSN-rendszerben a számítási és a tárolási kapacitás is szűk keresztmetszet, ezért fontos egy optimális algoritmus megtervezése.

A hangfeldolgozáshoz először **digitalizálni** kell a hangot, majd a kapott digitális adatfolyamon el kell végezni a **hangelemzést**. A két fázis egymástól nem független, hiszen a digitalizálás paramétereit úgy kell beállítani, hogy az a hangelemzést végző algoritmusoknak megfeleljen. Ezért a fejezet további részeiben először a digitalizálás folyamatáról és paramétereiről lesz szó általánosságban, majd a dolgozat szempontjából releváns algoritmusokról, végül a fejezet újra a digitalizálással fog záródni, de ezúttal célirányosan, a konkrét alkalmazáshoz szükséges paraméterek megtalálása lesz bemutatva szimulációkkal és példákkal.

5.2.1. ADC-kről általában

A minket körülvevő világban minden folyamat analóg módon megy végbe, még a legtöbb számítógép és mikrovezérlő csak digitális formában tud dolgozni az információval. Ezért ha valamilyen fizikai paraméterrel kell dolgozni, akkor a számítások elvégzéséhez az analóg értéket át kell alakítani digitálissá. Ezt a feladatot látják el az analóg-digitális konverterek, vagyis (a szakirodalomban használt angol rövidítéssel) az ADC-k.

A 13. ábrán látható ADC három logikai egységre bontható:

- 1.) **Analóg érzékelő.** Ebben az esetben egy mikrofon.



13. ábra. Egy ADC elvi felépítése

- 2.) **Mintavevő.** Az analóg jelben először az idő folytonosságát kell megszüntetni. Ehhez rögzített T_s mintavételi idővel mintát kell venni a jelből.
- 3.) **Kvantáló.** Végül az időben diszkrét jelet értékkészletben is diszkrétizálni kell. Ez a kvantáló feladata. A kvantálásnak két lényeges paramétere van: a kvantálási fokok száma (bitmélység) és a kvantáló típusa. A továbbiakban csak a bitmélység optimalizálásáról lesz szó, a kvantáló típusa lineáris lesz. (Ennek indoklása az 5.2.3. fejezetben megtalálható.)

A mintavételi frekvencia ($f_s = \frac{1}{T_s}$) kiválasztásakor négy szempontot kell figyelembe venni:

- a rögzíteni kívánt maximális frekvenciát (f_{max});
- a hangot feldolgozó ADC sebességét (ADC_{max});
- a tárolókapacitást;
- a számítási kapacitást.

Az ADC sebessége a használt technológiától, kvantálási szintektől és az ártól függően széles skálán mozog. Ennek több oka van. Egyrészt a kvantáló gyakran kondenzátorokkal működik, amiket fel kell tölteni. Másrészt a mintavevőben lévő kvarcnak van egy bizonytalansága, ami nagy jittert okoz, ha a mintavételi idővel egy nagyságrendbe esik. Az ADC maximális sebessége megtalálható az eszköz adatlapján. Egy PIC18F14K22-es mikrovezérlő ADC-je 10 bites minták esetén másodpercenként ~ 2000 mintát tud rögzíteni. A mai hangkártyák akár 96kHz-es mintavételi sebességre is képesek 24 bites kvantálás mellett.

Tehát ha adott a feldolgozni kívánt maximális frekvencia és az ADC, akkor van egy szigorú alsó és felső határ:

$$2 \cdot f_{max} < f_s \quad (1)$$

$$f_s < ADC_{max} \quad (2)$$

Így általában f_s -re egy széles tartomány adódik, amit a rendelkezésre álló tároló és számítási kapacitás tud jobban behatárolni. Minél kisebb f_s értéke, annál kevesebb memória kell a minta tárolásához, ugyanakkor minél nagyobb f_s értéke, annál kevesebb nagyfrekvenciás ($f > \frac{f_s}{2}$) zaj lapolódik vissza a vizsgált tartományra [27]. (A nagyfrekvenciás komponensek visszalapolódását a 7.2.4. mérés mutatja be.)

A digitalizálás utolsó lépése a kvantálás. Az eredeti jel (feltéve ha Shannon mintavételi törvénye teljesül) eddig a lépésig visszaállítható, ezután viszont nem. Lineáris kvantálót

feltételezve a kvantálás egyetlen paramétere a bitmélység, vagyis az, hogy hány biten történik egy analóg érték ábrázolása. Minél nagyobb ez az érték, annál jobb lesz a hang minősége (kisebb lesz a kvantálási zaj), de több memóriát és számítási kapacitást is igényel a minta tárolása.

Az 5. táblázat felsorol néhány hétköznapi alkalmazást és a hozzájuk tartozó ADC-paramétereket:

5. táblázat. *Különböző hétköznapi alkalmazások kvantálási paraméterei [26]*

F_s	Bitmélység	Alkalmazás
7kHz	8 bit	Mobiltelefon
22.05kHz	8 bit	Középhullámú rádió
44.1kHz	8 bit	CD minőség
48kHz	24 bit	fél-professzionális stúdió
96kHz	24 bit	DVD - Audio
192kHz	32 bit	professzionális stúdió

5.2.2. A vett hagsminták feldolgozása

Az alkalmazás feladata, hogy nagy zaj mellett is felismerjen egy sípoló hangot, amit egy mobiltelefon ad ki. Bár a mobiltelefonok képesek tisztán és folytonosan kiadni egy adott frekvenciájú szinuszos hangot, a rendszert nem ajánlott egyetlen egy frekvenciára alapozni. (Lásd 7.1. fejezet) A ténylegesen detektálandó hangnak frekvenciamoduláltnak kell lennie, például 400Hz és 500Hz között szinuszosan kell változnia. Így sokkal kisebb a valószínűsége annak, hogy valamilyen zajforrás vagy a fading elfedi a jelet, ugyanakkor komplexebb jelfeldolgozó algoritmusok szükségesek a felismeréshez.

Ez a fejezet egy olyan algoritmust mutat be, ami a vett hangminták alapján eldönti, hogy található-e a közelben megfelelő hangforrás. Az algoritmus bemenete a digitalizált hang, kimenete pedig egy valószínűségnek feleltethető meg.

Áttérés a frekvenciatartományba

Bár a hang felvételekor egy időtartománybeli jelet ad ki az ADC, a hangfeldolgozást sokkal egyszerűbb a frekvenciatartományban elvégezni, ahová analóg jelek esetén a Fourier-transzformációval (3) lehet áttérni.

$$X(\xi) = \int_{-\infty}^{\infty} x(t) e^{-2\pi \cdot i \cdot t \cdot \xi} dt. \quad (3)$$

A 3. egyenletet ebben a feladatban két ok miatt nem lehet használni: I) a feldolgozandó jelek nem folytonosak (mintavettek) ill. II) A frekvenciák nem állandóak (ablakozni is kell).

Mindkét problémára megoldást ad a diszkrét Fourier-transzformáció (DFT), melynek feladata, hogy véges (N -hosszú) ablakokban keressen frekvenciakomponenseket:

$$X_n = \sum_{k=0}^{N-1} x_k \cdot e^{-2\pi i n k / N} \quad (4)$$

A DFT segítségével már meg lehet mondani a véges, mintavett jelről, hogy milyen frek-

venciakomponenseket tartalmaz [28], de a tényleges jeldetektáláshoz ez az információ még nem biztos, hogy elég. A probléma az, hogy a DFT minden frekvenciakomponensre csak egy átlagot fog adni és nem képes megjeleníteni a frekvenciakomponensek időbeli változását, ill. túl érzékeny a zajra. (Egy adott frekvenciatartományban folyamatosan jelenlévő, gyengae zaj elfedheti a hasznos jelet.) Ez sajnos alkalmatlanná teszi a detekció elvégzésére. Az erre vonatkozó mérések megtalálhatók a 7.2.1. fejezetben.

Rövid idejű Fourier-transzformáció

Arra, hogy a Fourier-transzformációban nem lehet megfigyelni az egyes frekvenciák időbeli változását, megoldást nyújt a rövid idejű Fourier-transzformáció [29] (továbbiakban STFT), ami az egyes frekvenciakomponensek intenzitását az idő függvényében ábrázolja.

Bemenete a digitalizált hang, kimenete pedig egy szürkeárnyaltos kép (kétdimenziós függvény), ahol konvenció szerint az idő függőleges irányban lefelé, a frekvencia pedig vízszintes irányban balról jobbra növekszik. Tehát az 1000. időegységben a 420Hz-es komponens teljesítménye a (420,1000) koordinátán lévő pixel intenzitásával arányos.

A STFT-vel fel lehet dolgozni azokat a mintákat is, amik az DFT-vel értelmezhetetlenek. (A többi mintára pedig sokkal egyértelműbb kimenetet ad.) Az erre vonatkozó mérések a 7.2.2. fejezetben láthatóak. Az új probléma az, hogy az STFT kimenete egy szürkeárnyaltos kép, amit nehéz elemezni.

Hatékony képelemzés

A képelemzés, vagy mintázatfelismerés jellemzően számításigényes feladat. Rengeteg algoritmus és protokoll lett kifejlesztve, hogy a képek elemzése minél megbízhatóbb legyen. Ezeknek az algoritmusoknak közös jellemzője, hogy nagy a számítási igényük, ezért nem lehet implementálni egy WSN-ben.

Ebben a szakaszban arról lesz szó, hogy a STFT-től kapott képet hogyan lehet mégis néhány egyszerű lépésben kielemezni. Az algoritmus keresése során az alábbi két szempontot kell figyelembe venni:

- Nincs szükség 100%-os pontosságra, mivel a szenzorok folyamatosan mérnek. Néhány téves eredmény nem okoz gondot.
- Nem szükséges a teljes spektrogramot kielemezni, mert az eredmény szempontjából csak a 400Hz és 500Hz közötti tartomány érdekes, annak is csak egy másodperces szakasza.

A továbbiakban a fentieket figyelembe véve egy olyan képfelismerő algoritmus kerül bemutatásra, ami nagy pontossággal képes detektálni az 1Hz-es szinuszhullám egy periódusát. Az algoritmus működése itt csak ismertetve lesz, de a 7.2.3. fejezet bemutatja, hogy hogyan milyen kimenetet ad az eddig elemzett hangmintákon.

Az algoritmus lépései a következők:

1.) Szükséges részletek kivágása:

az eredmény szempontjából az eredeti képnek csak a 400Hz és 500Hz közötti 1 másodperces mintájára van szükség.

2.) **Kép binárisá alakítása:**

a bináris képpel sokkal könnyebb dolgozni, ennek a lépései a következők:

- (a) Legvilágosabb pixel megkeresése. Ez az érték legyen E_{max}
(A pixelhez tartozó komponens energiája a pixel intenzitásával arányos.)
- (b) A bináris kép kialakítása:

$$\forall(x, y) \ E(x, y) < \frac{E_{max}}{c} \rightarrow E(x, y) = 0$$

$$\forall(x, y) \ E(x, y) \geq \frac{E_{max}}{c} \rightarrow E(x, y) = 1$$

ahol $E(x, y)$ az (x, y) koordinátájú pont intenzitása. A $\frac{E_{max}}{c}$ pedig egy küszöbérték, ahol c egy valós szám. Ezt tapasztalat útján lehet meghatározni.

- 3.) Ha a kapott képen a pixelek 10%-25%-a $E(x, y) = 1$ -es, akkor a vett hangmintát el kell fogadni, ellenkező esetben el kell utasítani. (Ez a százalék könnyen átalakítható valószínűséggé is.)

Az algoritmus hátránya, hogy csak a pixelek egymáshoz viszonyított intenzitásával tördik, a tényleges elrendezésükkel nem. Lehetséges ennél sokkal zajtűrőbb algoritmusok létrehozása is, amelyek figyelembe veszik a zajok jellemző mintázatait, a keresett jel pontos formáját vagy akár az előző méréseket is. Ekkor az eredmény ugyan pontosabb, de nagymértékben növekedne a számítási- és memóriaigény. A fentiekben ismertetett algoritmus nagyon gyors. Ha a feldolgozandó (kivágott, tehát csak a lényeges részeket tartalmazó) kép pixelszáma n , akkor az algoritmusnak

- el kell végeznie egy kivágást, ami megfelelő címaritmetikával elkerülhető,
- a kivágott részen el kell végeznie egy maximumkeresést, amihez n összehasonlítás és $\frac{n}{2}$ értékadás kell,
- binárisá kell alakítania a képet, ami újabb n összehasonlítást és n értékadást igényel,
- végül összegeznie kell a kép pixeleit, ami n összeadást jelent.

Mivel a legtöbb mikrovezérlő-architektúrán az összehasonlítás 2, az értékadás és az összeadás pedig 1-1 utasításciklus, ezért könnyen megadható az algoritmus futási ideje, ami csak n -től és az utasításciklus idejétől (t_c) függ:

$$t(n, t_c) = 7.5 \cdot n \cdot t_c \quad (5)$$

A mérések alapján n jellemzően 6000 körüli érték. Ez azt jelenti, hogy egy elterjedt PIC18F14K22-es mikrovezérlőn, amin $t_c = 1/4MHz = 250ns$, az algoritmus

$$t(6000, 250ns) = 7.5 \cdot 6000 \cdot 250ns = 11.25ms.$$

alatt képes lefutni.

5.2.3. Az ADC paramétereinek beállítása

A bemutatott algoritmus működését alapvetően a **mintavételezési frekvencia** (f_s) és a **bitmélység** (n) befolyásolja.

Először a mintavételi idő meghatározásáról lesz szó. Ez rendelkezik egy éles alsó és felső határral (5.2.1. fejezet)

A felső korlát a megvalósítás során alkalmazott mikrovezérlő ADC-jétől függ. A tényleges megvalósításról a 8. fejezetben lesz szó, ahol egy *AMB8423* típusú modul kerül majd bemutatásra, amelyben a mikrovezérlő (*MSP430F2274*) ADC-je 2kHz-es frekvenciával képes mintavételezni, ami megadja a felső korlátot:

$$f_s < 2kHz$$

A legnagyobb detektálendő frekvencia komponens 500Hz, amiből (Shannon - Kotelnikov mintavételi tétele alapján) következhetne, hogy a minimális mintavételi frekvencia 1000Hz legyen, azonban a vizsgálandó jelmintázat alulmintavételezhető (a detektálendő jel sávszélessége 100Hz):

$$400Hz < f_s$$

Az ezt igazoló mérés megtalálható a 7.2.4. fejezetben.

A fentiekből f_s -re egy 1600Hz-es tartomány adódik, de a pontos meghatározáshoz a bitmélységet is figyelembe kell venni. A bitmélység és az f_s együtt határozzák majd meg a minta tárolásához szükséges memóriát, ami WSN esetén kritikus kérdés.

Először a legnagyobb mintavételi frekvencia esetén (2kHz) kell meghatározni a minimális n bitmélységet, majd ehhez a bitmélységhez kell meghatározni a minimális mintavételi frekvenciát. Az erre vonatkozó méréseket a 7.2.5. fejezet tartalmazza. Ez alapján az algoritmus 600Hz-es mintavételi frekvencia és 1 bites kvantáló mellett is képes detektálni a keresett hangmintát.

Ezután meg kell vizsgálni, hogy érdemes-e növelni a bitmélységet, ha ezzel csökkenteni lehet a frekvenciát. Ahhoz, hogy a bitmélységet egyről kettőre lehessen növelni plusz memóriaigény nélkül, a mintavételi frekvenciát felezni kell. Ez azt jelenti, hogy kétbites kvantáló mellett 300Hz-es mintavételi frekvenciára (vagy ennél kisebbre) lenne szükség. A 7.2.4. fejezetben ismertetett mérésből viszont kiderül, hogy 400Hz alatti mintavételezés mellett már semmilyen szinusz-jellege nincs a kapott spektrogramnak, így a detektálása lehetetlen.

Tehát az ADC végleges paraméterei: i) Mintavételi frekvencia: 600Hz és ii) Bitmélység: 1 bit.

5.3. Az algoritmusok implementálása WSN-re

Sikerült megtervezni egy olyan algoritmust, ami képes kielemezni egy alacsony minőségű hangot, de még így is nagy a számításigénye és a memóriaszüksége. (A teljes hangfeldolgozó algoritmusba beletartozik a spektrogram legenerálása is.) Ebben a fejezetben arról lesz szó, hogy ezt az algoritmust hogyan lehet átültetni a gyakorlatba. Bár konkrét forráskódokról még itt nem lesz szó, a fejezet további része felfogható egy megvalósíthatósági tanulmányként.

5.3.1. A klasszikus WSN-architektúra

A klasszikus WSN-hálózat egy nagy számítási kapacitású nyelőből és több (esetenként több száz) csomópontból áll. A csomópontok feladata az adatok mérése és a csomagok továbbítása rádiós úton, minden más műveletet, így az adatfeldolgozást is, a nyelő végzi.

A legegyszerűbb megoldás az lenne, ha a nyelő begyűjtené a mért értékeket és minden hangmintára futtatná a hangfeldolgozó algoritmust. Ez sajnos nem megoldható.

A probléma bemutatása egy gyakorlati példán keresztül történik, melyben a feladat a budapesti Aréna Pláza biztosítása egy esetleges katasztrófa esetére.

A pláza alapterülete $380m \times 130m$ és négy szintes, vagyis a teljes terület

$$4 \cdot 380m \cdot 130m = 197600m^2$$

A 7.1. mérésből látszik, hogy egy olcsó mikrofon érzékelési sugara ideális esetben 3 méter, de romok között ez két méterre is csökkenhet, tehát egy szenzor által lefedett terület:

$$2 \cdot \pi \cdot 2 \approx 13m^2$$

Ebből az következik, hogy az Aréna pláza teljes lefedéséhez 15200 darab csomópont kell.

Minden csomópont folyamatosan mér 600Hz-es mintavételi frekvenciával és 1 bites kvantálással, ami azt jelenti, hogy 1 másodperc alatt 600 bitet, vagyis 75 bájtot továbbítanak a központ felé.

Ez összesen $15200 \cdot 75 = 1140000$ bájt, ami 1.1MB.

Egy esetleges omlás után nem szükséges, hogy minden csomópont folyamatosan mérjen, megengedhető, hogy csak minden 10. másodpercben történjen egy mérés. Ez $\sim 100kB$ -ra csökkenti a másodpercenként termelődő hasznos adatot, az adatforgalom tehát:

$$100KB/sec = 800Kbps$$

Ez azt jelenti, hogy a nyelő közelében a másodpercenkénti átlagos hasznos adatforgalom 800Kb lesz. Az alábbi táblázat néhány elterjedt vezeték nélküli protokoll és modul adatsebességét tartalmazza:

6. táblázat. Elterjedt WSN-rendszerek

Szabvány/modul neve	Hasznos adatsebesség
ZigBee	250Kbps
AMB8443	500Kbps
Bluetooth 1.2	700Kbps
Bluetooth 2.0	2.1Mbps

Ahogy a táblázatból is látszik, a szükséges adatsebességet legfeljebb a Bluetooth 2.0-ás verziója tudná biztosítani, de a Bluetooth alkalmatlan WSN-hez. Ráadásul az Aréna Pláza nem számít nagy épületnek. A feladat megoldására a klasszikus WSN-architektúra nem alkalmas.

5.3.2. Algoritmus futtatása a csomópontokon

Ha a hálózat képtelen eljuttatni a nyers mérési adatokat a nyelőkhöz, akkor kénytelen elvégezni a számítást lokálisan. A számítások elvégzéséhez először is el kell tárolni a mért hangmintát, meg kell határozni a spektrogramját (legalább a detektáláshoz szükséges részletét), végül futtatni kell a jeldetektáló algoritmust. A teljes folyamat memóriaigénye:

- 1.) 1 másodperces, egybites felvétel eltárolása: $600Hz \cdot 1s \cdot 1bit = 600bit = 75B$
- 2.) Spektrogram pontjainak eltárolása (legalább 8 biten): $6kB$

Tehát olyan mérőpontokra van szükség, amik rendelkeznek a kellő számítási kapacitással és memóriával. A mérés elvégzése során egy másodperc alatt rögzíteni kell 75 bájtnyi hangmintát, ebből le kell generálni legalább 60, egyenként 100 elemet tartalmazó DFT-t (a spektrogramhoz) és a binárisra történő átalakításig tárolnia is kell minimum 8 bites pontossággal, vagyis összesen $60 \cdot 100 + 75 = 6075$ bájtot kell tárolnia, ami csak a mérések elvégzéséhez kell. Ebben nincs bent a program és a többi információ sem, amit más okokból kell tárolni. Egy átlagos, olcsó mikrovezérlőnek nincsen ennyi használható memóriája, tehát a feladatra erősebb eszközöket kell keresni, például egy onboard számítót (továbbiakban OBC). Ma már kaphatóak nagyon olcsó és kis fogyasztású OBC-k, amik felérnek egy 5 évvel ezelőtti asztali számítógéppel. Egy ilyen képes lenne elvégezni a szükséges számításokat, a nyelőknek pedig csak akkor küldene egy rövid csomagot, ha talál valamit.

A feladatra a legjobb választásnak a Raspberry Pi tűnik, hála alacsony fogyasztásának (1.5W), alacsony árának (aktuális árfolyamtól függően 8000Ft körül mozog) és nagy teljesítményének. Ez az OBC 700MHz-es processzorral és 512MB RAM-mal rendelkezik, a háttértára pedig 32GB-ig bővíthető. Képes futtatni több fajta Linux-disztribúciót (például Debian-t és Arch Linux-ot) és rendelkezik 17 darab TTL-szintű GPIO-val (General Purpose Input/Output).

A gond az, hogy a megfelelő dobozzal, akkumulátorral és modulokkal együtt egy ilyen eszköz anyagára 60.000Ft lenne. Ebből gazdaságosan nem lehetne 15200 darabot telepíteni az Aréna Plázába. Meg lehetne próbálni növelni a mikrofon hatótávolságát, ezzel akár kétszeres vagy négyszeres hatótávolságot is el lehetne érni és így elég lenne ritkábban telepíteni. (Ez lehetséges, hiszen a Raspberry Pi képes futtatni a szükséges szűrőalgoritmusokat.) Ekkor viszont három probléma merül fel:

- egy csomópont sérülése sokkal nagyobb vakfoltot eredményezne;
- a ritkább telepítés miatt a WSN kommunikációs csatornái sokkal sérülékenyebbek lennének;
- mivel a Raspberry Pi nagyobb mint egy átlagos csomópont, sérülékenyebb is, tehát egy omlás következtében nagyobb valószínűséggel menne tönkre.

Ezek miatt nem tanácsos a ritkábban telepített, drágább csomópontok mellett dönteni, mert a kapott rendszer sérülékenyebb és megbízhatatlan lenne.

5.3.3. Jelfeldolgozás párhuzamos algoritmusokkal

Bár a nyelőig továbbra sem lehetséges eljuttatni a nyers mérési eredményeket, ettől még a csomópontok kihasználhatják a közelben lévők számítási és tárolási kapacitását. Az implementációs részben bemutatásra kerülő MSP430F2274 mikrovezérlőben 1kB belső memória van. A protokoll implementálása után ebből 700B marad a mérések elvégzésére.

A hangmintát (75B) le tudja tárolni, de a maradék memória már kevés arra, hogy kiszámolja a spektrogramot (6kB). Ehhez tízszer annyi memóriára lenne szüksége, mint amennyivel rendelkezik. A megoldás: a jelfeldolgozás során működjön együtt 10 csomópont. Együtt már képesek lennének tárolni a szükséges adatokat.

Egy párhuzamos környezetben futó alkalmazásnak a feladatait szét kell osztania térben és időben. A térbeli szétosztás azt jelenti, hogy az egyik csomópontnak támaszkodnia kell a többi csomópont memóriájára és számítási kapacitására, képesnek kell lennie távolról vezérelnie egy másik csomópontot. Az időbeli szétosztás egy kicsit bonyolultabb

fogalom. Mivel a csomópontoknak csak az idő kis szeletében kell annyi erőforrás, amivel egyedül nem rendelkeznek, ezért az ütemezést úgy kell szinkronizálni, hogy egyszerre csak egy csomópontnak legyen szüksége a többiek erőforrására. Ez a szinkronizálás jelenti a feladatok időbeli elosztását.

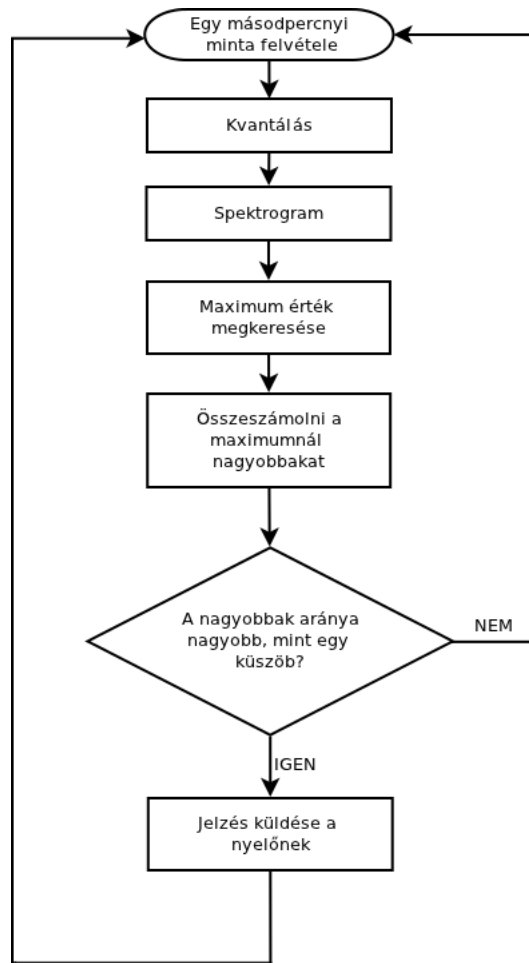
Az alkalmazás megtervezése előtt meg kell határozni, hogy milyen szolgáltatásokat vár el az alsóbb rétegektől ahhoz, hogy képes legyen a feladatokat elosztani és ne kelljen foglalkoznia azzal, hogy közben a többi csomópont is végzi a saját feladatát.

- A megosztott erőforrások (kooperáló node-ok) számának meghatározása: ez a szám futásidőben nem változhat, egyetlen egyszer, a program indulásakor kell megadni.
- Szabad memória megadása: Az alsó rétegeknek tudni kell, hogy az alkalmazásréteg mennyi memóriát igényel, amikor nem kell kooperálnia. A többit meg lehet osztani.
- Az alkalmazásrétegnek jeleznie kell, ha a program végrehajtása során egy olyan, védett helyre ért, ahol szükség van a többi csomópont erőforrására is. Mivel ilyenkor a node-ok közös erőforrásokat használnak, ezért ketten nem lehetnek egyszerre ilyen állapotban. Erről a klasszikus párhuzamos algoritmusok esetén szemafor gondoskodik. Ha egy csomópont megszerezte a szemafort, akkor:
 - adatokat tud küldeni a többi csomópontra;
 - adatokat tud visszaolvasni a többi csomópontról;
 - az átküldött adatokon különböző műveleteket tud elvégeztetni (távoli függvényhívás).
- Az alkalmazásrétegnek jeleznie kell, ha kilépett a védett szakaszból.
- Az alsóbb rétegeknek biztosítania kell, hogy ha egy csomópont nem engedi el a szemafort (például mert lemerül, vagy lefagy), a többi érzékelje és kezelje a hibát.

A fejezet további részeiben csak az alkalmazásrétegről lesz szó, ami feltételezi a fenti funkciók meglétét. Az elosztott számításokat támogató protokoll és API pontos megvalósítását a 6. fejezet mutatja be. A jelfeldolgozás futtatásához a 14. ábrán látható folyamatot kell elvégezni.

Ezeknek a lépéseknek a nagy része párhuzamosítható:

- **Hang felvétele:** a hang felvétele tölti ki az idő nagy részét (1s) de csak minimálisan veszi igénybe a csomópontot. Nem lehet és nem is érdemes párhuzamosítani.
- **A spektrogram adott részének kiszámítása:** a spektrogram kiszámításához el kell végezni 20 részleges FFT-t 30 frekvenciára és tárolni kell a keletkezett 3kB-ot. Ezekre egy csomópont alkalmatlan, ezért itt már szét kell osztani a feladatokat. Dolgozzon együtt egyszerre 10 csomópont. Ekkor mindegyiknek el kell végeznie egy FFT-t és 300B-t kell csak tárolnia, ami nem okoz gondot, két FFT pedig rövid idő alatt elvégeztethető egy mikrovezérlővel [30].
- **Maximum érték megkeresése:** a maximum érték megkeresése nem nehéz feladat, de a minta nincs meg egyben sehol, ezért ezt is párhuzamosítani kell. Mindegyik csomópont megkeresi, hogy a saját mintájában melyik a legnagyobb érték, majd a kapott eredményt elküldi a mérést végző csomópontnak. A kapott eredményekből már meg tudja tudni keresni a maximumot.



14. ábra. A hangelemzés lépései a felvételtől a jeldetektálásig

- **Az E_{max} maximum értékből meghatározott küszöbnél nagyobb értékek összeszámolása:** a mérést végző csomópont mindenkinek elküldi a küszöbértéket, ahol az egyes csomópontok összeszámolják az annál nagyobb értékeket. A kapott eredményt visszaküldik a mérést végző csomópontnak, ahol megtörténik a tényleges összeszámolás.
- **Az üzenet elküldése a nyelőbe (ha szükséges):** végül a mérést ténylegesen elvégző (és a párhuzamos számítást vezérlő) csomópont elküldi a nyelőnek a kapott számot, ha az egy meghatározott tartományban van. Ekkor már nem kellene sem az eddigi értékek, sem a többi csomópont segítsége.

Ahogy a fentiek alapján látható, az egyetlen számításigényes algoritmus a STFT, de az felbontható több, egymástól független FFT-re, amik a mikrovezérlőkkel már elvégezhetőek. A feladat a párhuzamos architektúrával megoldható. Azt is érdemes észrevenni, hogy az egyes csomópontok az idő nagy részében csak mérnek. (Egy másodpercnnyi mérésre jut pár ezredmásodperces feldolgozás, amihez szükség van a többi csomópontra is.) Ez azt jelenti, hogy az összes csomópont tud folyamatosan mérni.

6. Elosztott számításokat támogató protokoll

Elosztott algoritmusok alatt olyan algoritmusokat ért a szakirodalom, melyek képesek több, egymással valamilyen módon összekapcsolt feldolgozóegységen futni. Az algoritmusok egyes részei egyidejűleg, egymástól függetlenül hajtódnak végre, de bizonyos esetekben szükség lehet a folyamatok szinkronizálására és az egyes feldolgozóegységek közötti adatcserére. Az algoritmusok tervezése során az egyik legnagyobb kihívás az, hogy a rendszernek akkor is működnie kell, ha annak egyes részei (vagy a részegységek közötti kommunikáció) hibás [31].

Ebben a fejezetben egy olyan protokoll kerül bemutatásra, ami képes több csomópontot összekapcsolni és párhuzamos algoritmusokat futtatni. Először a protokoll pontos feladatait kell tisztázni:

- **Számítási klaszterek hatékony összeállítása:** A csomópontok aktiválás utáni első feladata, hogy összeállítsák a megfelelő méretű klasztereket, amiken belül megfelelő sebességgel és rövid késleltetéssel lehet kommunikálni.
- **Párhuzamos számítások támogatása:** A protokollnak hatékonyan kell támogatni az adatátvitelt és a távoli eljáráshívást a klasztereken belül.
- **Klaszterek dinamikus újraszervezése:** Egy csomópont meghibásodása esetén újra kell szervezni a klasztert.
- **Mozgó nyelők kiszolgálása:** A hálózatnak az alkalmazás sajátossága miatt egy olyan, klaszterektől független útvonalválasztást kell alkalmaznia, ami hatékonyan tudja kezelni a mozgó nyelőket.

A fejezetben egy, a fentieket támogató protokoll kerül bemutatásra az OSI-modell mentén haladva. (Az OSI-modellről már volt szó a 3.2.2. fejezetben.)

6.1. Fizikai réteg

A fizikai réteg feladata, hogy biztosítsa a közeghez történő hozzáférést. Ez magában foglalja az ütközések elkerülését, a csatorna effektív kihasználását és a modulok címezését is.

A kereskedelembe kapható moduloknak csak egy kis része enged hozzáférést a fizikai réteghez, rendszerint ezek sem teljeset. Az implementálásnál használt CC1101 típusú RF-chipben be lehet állítani a modulációs sémát, engedélyezni lehet a Manchester-kódolást és szabályozni lehet az adóteljesítményt. Ezeknek a paramétereknek a megfelelő hangolásával el lehet érni az optimális adatátviteli sebességet és a minimális fogyasztást, de ez nem része a dolgozatnak.

A fizikai rétegben lehet meghatározni a modulok címezésének módját is. A fejezet elején bemutatott igényeknek egy háromlépcsős címezési mód felel meg a legjobban. Minden modult három címen keresztül lehet elérni:

- 1.) Broadcast-cím: a broadcast-címre küldött adatokat minden modul megkapja. Ilyen üzenetek kellenek pl. a klaszterek felépítéséhez.
- 2.) Klaszter-szintű broadcast-cím: minden klaszternek kell egy egyedi cím. Az erre a címre küldött csomagokat minden modul megkapja, aki tagja a klaszternek. Ilyen

üzenetek kellenek ahhoz, hogy a node-ok szinkronizálhassák a működésüket a klaszteren belül.

- 3.) Egyedi modulcím: az erre a címre küldött üzenetet csak egy konkrét modul kapja meg. Ezek a címek a modulok közti kommunikációhoz kellenek.

6.2. Adatkapcsolati réteg

Az adatkapcsolati réteg biztosítja, hogy a csomagok két modul között ne vesszenek el. A réteg pontos megvalósításáról az implementáció során lesz szó, de a párhuzamos algoritmusok támogatását nem érinti.

6.3. Hálózati réteg

Ebben a protokollban egy node egyszerre két, logikailag elkülönülő hálózatban van jelen: egyrészt a saját klaszterében, másrészt egy, a klaszterezéstől független globális hálózatban. A hálózati rétegnek ezt a felosztást tükröznie kell, vagyis két, egymástól független modulból kell állnia.

6.3.1. Klaszterezés

A hálózat alapvető egységei a klaszterek. Ezek több, összekapcsolódott csomópontból állnak és topológiaiilag összetartozó egységet alkotnak, és az egy klaszteren belüli csomópontok hozzáférhetnek egymás erőforrásaihoz. Fontos, hogy a klaszterek felépítése optimális legyen.

A klaszterek felépítésének menete

A hálózat aktiválása után a csomópontok felépítik az M tagból álló klasztereket. (M értéke a működés során állandó.) Ennek menete a 15. ábrán látható.

A klaszterek felépítése során a csomópontoknak meg kell egyezniük egy ID-ben, ami a klasztert azonosítja és meg kell győződni arról, hogy a klaszterben megvan a szükséges létszám. Ezeknek a megállapításához arra van szükség, hogy az egymástól független folyamatok valamilyen megállapodásra jussanak egymás között, vagyis válasszanak egy vezetőt. Ez az osztott algoritmusok szakirodalmában jól ismert feladat.

A vezetőválasztás során adott több, egymástól független folyamat. Mindegyik rendelkezik egy saját egyedi azonosítóval és képesek egymással kommunikálni valamilyen topológia szerint. A vezetőválasztó algoritmus lefutása után pontosan egy folyamat választhatja magát vezetőnek [32], ami aztán megkezdheti felépíteni a klasztert.

Több vezetőválasztó algoritmus létezik, más idő- és üzenetigényekkel. Mivel vezeték nélküli hálózatok esetén a csatornkapacitás egy nagyon korlátozott erőforrás, ezért olyan algoritmusra van szükség, ami az üzenetek számát minél alacsonyabban tartja. Persze az is fontos, hogy az algoritmus egy adott időkomplexitással lefusson. Bebizonyítható [33], hogy ha az algoritmus lefutási idejének felső korlátja polinomiális, akkor az üzenetek számának alsó korlátja

$$\Omega(n \log n),$$

ahol n a hálózatban lévő összes csomópont száma.

A 15. ábrán látható algoritmus üzenetkomplexitásának felső korlátja $O(\frac{n^2}{M})$, ahol n az összes csomópont száma, M pedig a klaszterek mérete. Ez legrosszabb esetben négyzesen függ a hálózat méretétől, ráadásul mivel $n \gg M$, ezért ez $O(n^2)$ lesz. Az algoritmus elvetése előtt érdemes megvizsgálni, hogy a legrosszabb eset mennyire gyakran fordul elő, egyáltalán előfordulhat-e. Ehhez meg kell nézni, hogy miért ilyen sok az üzenetváltás.

A klaszterépítés folyamán háromféle üzenet jöhet létre:

- 1.) *CLUSTERBUILD*: Ha egy csomópont megpróbál klasztert építeni, akkor azt ezzel az üzenettel hozza a szomszédai tudtára. Minden klaszter felépülését ez az üzenet előzi meg, ezért ebből minimum $\frac{n}{M}$, maximum n generálódik. (A minimum akkor keletkezik, ha a minden üzenet sikeresen felépít egy klasztert.) Mivel $n \gg M$, ezért *CLUSTERBUILD*-üzenetből $O(n)$ darab keletkezik.
- 2.) *IAMREADYTOJOIN*: Egy *CLUSTERBUILD*-üzenetre minden csomópont egy *IAMREADYTOJOIN*-üzenettel válaszol. Legrosszabb esetben minden csomópont hall mindenkit, ezért minden *CLUSTERBUILD*-üzenetre az összes, addig szabad csomópont válaszol. Ha mindenki hall mindenkit, jogos a feltételezés, miszerint az összes klaszterépítő üzenet sikeres, tehát összesen $\frac{n}{M}$ *CLUSTERBUILD*-üzenet hangzik el. (Mivel $n \gg M$ miatt $O(\frac{n^2}{M}) = O(n^2)$, ez a feltételezés egyébként sem változtat az eredményen.) Ekkor az *IAMREADYTOJOIN*-üzenetek száma:

$$\sum_{i=0}^{\frac{n}{M}} n - i \cdot M = \frac{n^2}{M} - \frac{n}{2} = O(\frac{n^2}{M})$$

- 3.) *INVITE*: Mivel minden csomópont legfeljebb egyszer kap *INVITE*-üzenetet, ezért ezeknek az üzeneteknek a száma maximum n .

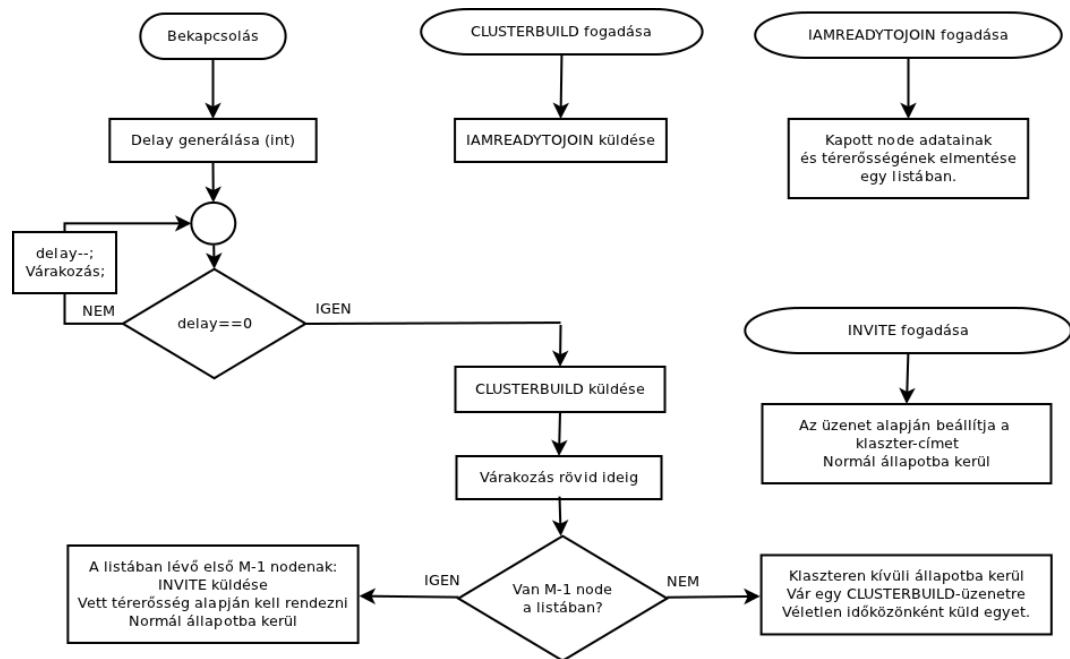
Az üzenetek számának nagyságrendjét az *IAMREADYTOJOIN*-üzenetek határozzák meg, ugyanis legrosszabb esetben arra minden csomópont válaszol. Ez azonban a konkrét alkalmazásban nem történhet meg. Egy csomópont körülbelül 20 méterre képes elküldeni az üzenetet a romok alatt, a szenzora pedig 2 méteres sugarú kört képes lefedni. Ez azt jelenti, hogy egy húsz méter sugarú körben

$$\frac{\pi \cdot 20^2}{\pi \cdot 2^2} = 100$$

csomópont van, vagyis egy *CLUSTERBUILD*-üzenetre legfeljebb ennyi *IAMREADYTOJOIN*-üzenet érkezik. Tízelemű klasztereket feltételezve ez a klaszterméret tízszerese. Tehát a ténylegesen elküldött üzenetek száma:

- 1.) $\sum CLUSTERBUILD = O(n)$
- 2.) $\sum IAMREADYTOJOIN = O(n \cdot M \cdot 10) = O(n)$
- 3.) $\sum INVITE = O(n)$

A fentiekből következik, hogy a klaszterépítő algoritmus üzenetkomplexitása lineárisan függ a hálózat méretétől. (Mivel az algoritmus kihasználja azt, hogy a broadcast-üzeneteket csak a hálózatnak egy kis része kapja meg, ezért nem áll ellentétben azzal, hogy egy polinomiális idő alatt lefutó vezetőségválasztáshoz legalább $O(n \cdot \log n)$ üzenet kell.)



15. ábra. A klaszterek felépítésének protokollja

A klaszterek építése négy, egymástól független folyamattal írható le. Az első folyamatot a bekapcsolás (vagy valamilyen más, külső esemény) indítja el, és az a feladata, hogy megpróbáljon felépíteni egy klasztert. Ehhez generálnia kell egy (a hálózat méretétől függő véletlen, vagy az egyedi azonosítóból képzett) számot, amiről visszaszámol. A visszaszámlálás végén kiküld egy *CLUSTERBUILD* broadcast-üzenetet, amit mindenki megkap vevőtávolságon belül, ezután vár egy kis időt. A várakozás végén ellenőrzi, hogy kik válaszoltak erre egy *IAMREADYTOJOIN* üzenettel. Ha legalább $M - 1$ csomópont válaszolt, felépíti a klasztert. A klaszter tagjai megkezdik a normál működést és a továbbiakban nem foglalkoznak a klaszterépítő üzenetekkel.

CLUSTERBUILD fogadása esetén csak válaszolni kell rá egy *IAMREADYTOJOIN*-nal. Ezt a *CLUSTERBUILD* feladója kapja meg.

IAMREADYTOJOIN fogadása esetén a küldő node címét és térerősségét le kell tárolni egy listában.

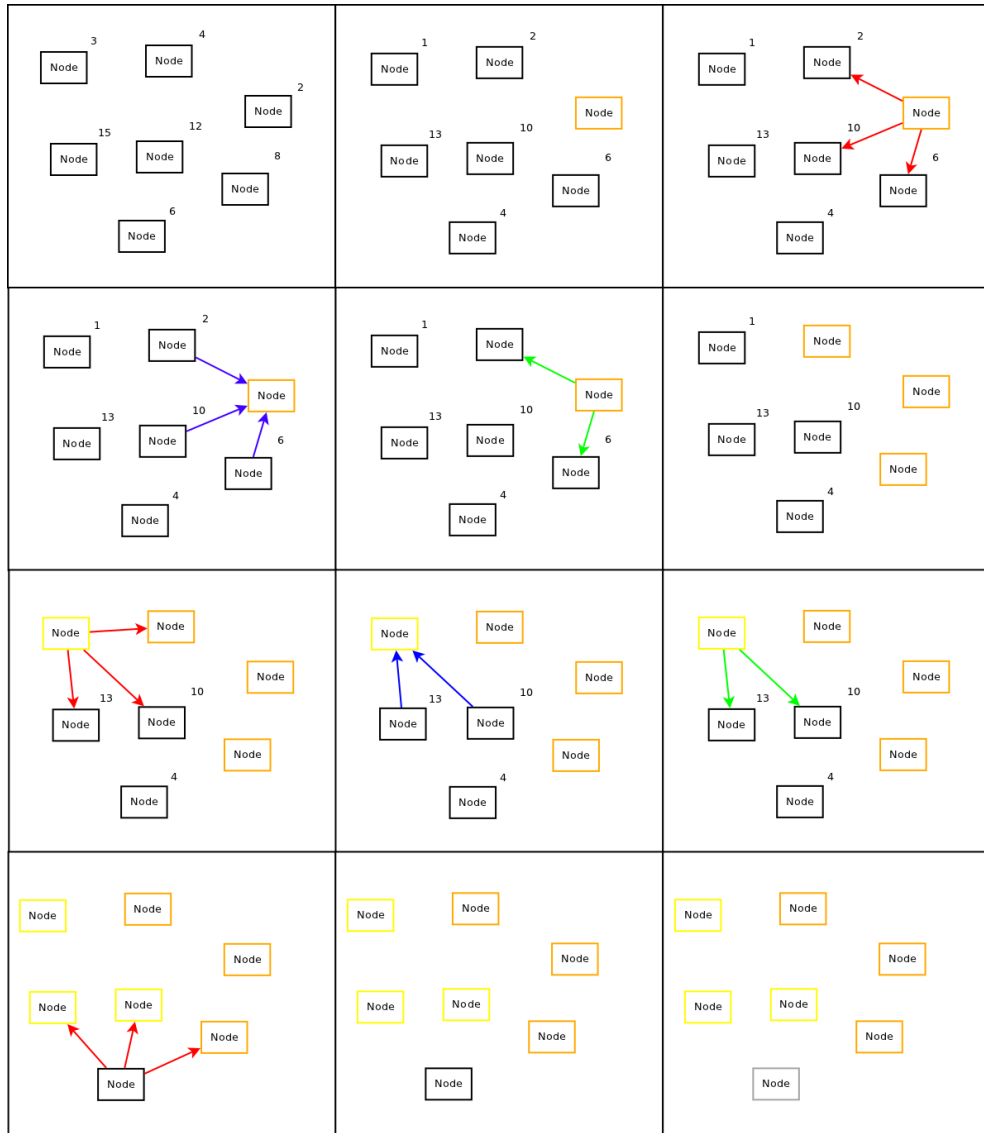
INVITE fogadása esetén be kell állítani a klaszter-címet (ez is az üzenettel együtt jön) és meg lehet kezdeni a normál működést.

Az időkomplexitás sem elhanyagolható kérdés, de könnyű belátni, hogy az is csak lineárisan függ a hálózat méretétől, ezért az itt bemutatott algoritmus a feladat szempontjából optimális.

A 16. ábrán látható egy lehetséges klaszterezés. Miután a klaszterek létrejöttek, előfordulhat, hogy vannak olyan node-ok, amik nem kerültek be egy klaszterbe sem. Ezek csak a nyelőkkel történő kommunikációban vesznek részt, de véletlen időközönként megpróbálnak klasztert építeni.

Klasztereken belüli útvonaltervezés

A klasztereken belüli hatékony és gyors kommunikáció nagyon fontos. Bár a klaszter felépítése miatt van legalább egy olyan csomópont, ami az összes többit látja, de nem biztos, hogy ez az összesre igaz. Szükség van egy olyan protokollra, amivel a klasztereken belül energiahatékony és gyors kommunikációt lehet biztosítani. A feladatot egyszerűsíti, hogy a klaszterekben viszonylag kevés csomópont van, és hogy az alkalmazás miatt a



16. ábra. $M = 3$ esetén a csomópontok egy lehetséges klaszterezése

topológia nem változik. (Ha pedig mégis, akkor a klaszterezést is újra kell kezdeni.)

Ilyen körülmények között egy útvonal táblázatok segítségével megvalósított útválasztó protokoll képes a leghatékonyabban működni. A táblázatok felépítése és tárolása könnyű, hiszen egy klaszter viszonylag kevés csomópontot tartalmazhat, amik maximálisan két hopnyi távolságra vannak egymástól.

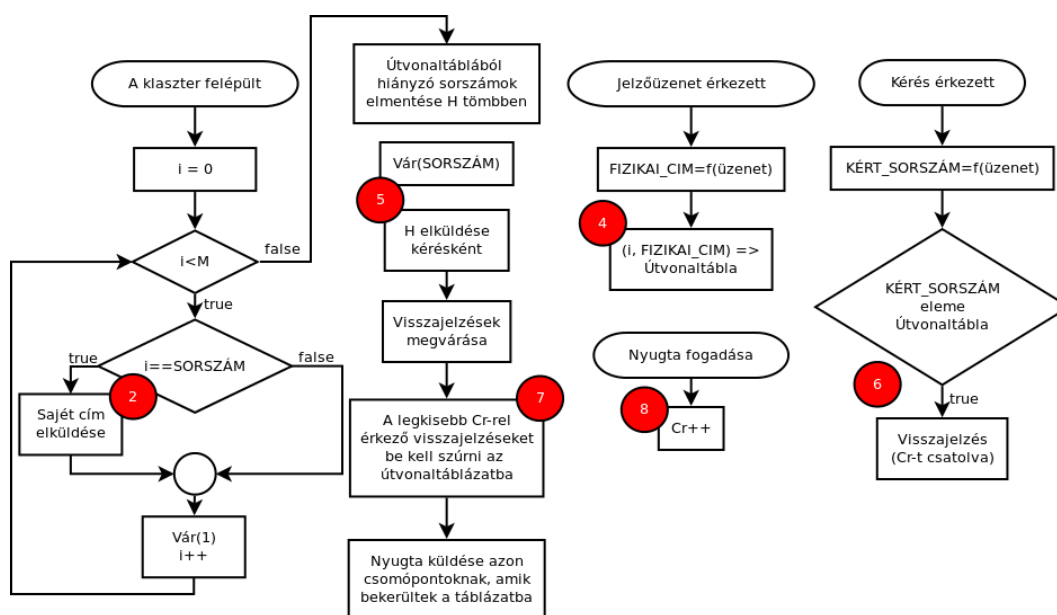
A klaszterek felépülése után az útvonal táblázatokat az alábbi lépésekkel lehet végrehajtani:

- 1.) A klaszter létrejötte után minden csomópont kap egy sorszámot 1 és M között.
 - Ezt a klaszterépítő csomópont ki tudja osztani
- 2.) Az összes csomópont a sorszámanak megfelelő ideig vár, majd elküld egy broadcast-üzenetet, amelyben közli a saját fizikai címét.
 - Ha a csomópontok (közel) egyszerre küldik el az üzeneteiket, akkor a mérések alapján rádiós modul sikertelenül alkalmazza a CSMA/CS módszert és előfordul, hogy az üzenetek ütköznek. Ezt a hibát az adatkapcsolati réteg unicast

esetén képes kijavítani azzal, hogy visszaigazolást vár az elküldött üzenetről, de broadcast esetén ez túl nagy terhet róna a hálózatra. Ezért kell itt TDMA-t használni.

- 3.) A csomópontok tudják, hogy melyik időrésben hányadik eszközt kell hallani. Ha valamelyik időrésben nem kapnak üzenetet, akkor az ahhoz tartozó csomóponttal nem tudnak közvetlenül kommunikálni.
- 4.) A hallott üzenetekhez tartozó csomópontokat beírják a saját routing táblájukba.
- 5.) Miután az egy hopnyi kapcsolatok megvannak, a már megkapott sorszámoknak megfelelő sorrendben mindegyik csomópont küld egy broadcast-üzenetet, ami az általuk elérhető modulok azonosítóját tartalmazza.
- 6.) Ezekre a broadcast-üzenetekre azok válaszolnak, amik az üzenetben lévő egyik csomópontot elérik.
 - Ezek a unicast üzenetek egyrészt tartalmazzák, hogy a kért csomópontok közül melyikeket érik el.
 - Másrészt azt, hogy eddig hány csomópontnak biztosítanak átjátszót. Ez a szám legyen a továbbiakban C_r .
- 7.) A válaszok segítségével meg lehet határozni, hogy az útvonaltáblában a keresett címzetthez melyik, közvetlenül elérhető csomópont kerüljön.
 - Több lehetőség esetén azt a csomópontot fogja választani átjátszónak, aminek a C_r -je a legkisebb. Ezzel a módszerrel el lehet osztani a klaszteren belül az útválasztásból adódó terhelést.
- 8.) Végül a kiválasztott csomópontokkal tudatni kell, hogy a továbbiakban átjátszóként viselkednek. Erre azért van szükség, hogy növelhessék a C_r -t.

A fenti algoritmus folyamatábrája a 17. ábrán látható. Az ott lévő sorszámok a fenti pontoknak felelnek meg.



17. ábra. Az útvonaltáblázatok felépülése

Az algoritmus lefutása után létrejövő útvonaltáblázatok segítségével a klaszteren belül a csomópontok energiahatékonyan képesek kommunikálni egymással. A protokolltáblázatok használatához minden csomagot csak a tényleges címzett címével kell kiegészíteni, ez alapján már el lehet juttatni minden csomagot a célba.

A klaszteren belüli üzenetküldés az adatkapcsolati réteg szolgáltatásaira épít, ami biztosítja, hogy a csomag nem veszt el két csomópont között. Ez sajnos még nem garantálja azt, hogy az üzenet valóban célba ér, hiszen lehetséges, hogy az első hop megtörténik, de a második már nem. Ez esetben a hibát jelezni kell az eredeti feladónak. Ezt két módszerrel lehet megvalósítani. Vagy a címzett visszaigazolja a fogadott csomagot, vagy csak hiba esetén (a hiba helyétől) kap egy értesítést. Az utóbbi megoldás mellett két érv szól:

- A klaszter felépítése miatt két csomópont maximális távolsága két hop. Az üzenet kézbesítése így vagy a feladónál hiúsul meg (amiről azonnali értesítést kap), vagy a közvetlen szomszédságában, ahonnan szintén egyszerű visszaüzeni a hiba tényét (esetleg okát).
- Mivel az üzenetek nagy része sikeresen célba ér, (ha két csomópont nem tud egymásnak üzeni, bontani kell a klasztert) érdemesebb a kudarcról üzeni. (Egy hibajelzésnek sokkal nagyobb az információértéke, mint egy visszaigazolásnak.)

Klaszterek felbomlása és újraszerveződése

Előfordulhat, hogy az egyik node lemerül vagy elromlik. Ez esetben a klaszter működésképtelenné válik. Ilyenkor meg kell próbálni újra felépíteni egy klasztert.

A klaszterek újraszervezésének menete:

- 1.) Az M tagú klaszter egyik tagja meghibásodik.
- 2.) A többi csomópont érzékeli a hibát, felbontják a klasztert.
- 3.) Innentől mindegyik klaszteren kívüli csomópont lesz, melyek meghatározott időnként megpróbálnak felépíteni egy klasztert.

Fontos kitérni arra, hogy a csomópontok hogyan érzékelik a klaszterek felbomlását. A legkézenfekvőbb megoldás, hogy a hálózati réteg adott időnként megpróbál kapcsolatba lépni a klaszter többi tagjával. Ha ez nem sikerül neki, akkor a klaszter felbomlott. Ilyenkor meg kell szakítani az alkalmazásréteg futását és át kell váltani klaszteren kívüli üzemmódba. A megoldás előnye, hogy egyszerű megvalósítani és az alkalmazásrétegtől semmilyen közreműködést nem igényel. Hátránya viszont, hogy az alkalmazásrétegnek fel kell készülnie arra, hogy bármelyik utasítás után megszakadhat a működése. Ez lehetetlenné teszi azt, hogy egy védett tranzakciót végrehajtsa. (Például hogy az EEPROM-ban módosítson egy több szegmensből álló értéket, aminek fontos a konzisztenciája.) Ez egy általános protokolltól túl szigorú megkötés lenne, ezért más megoldás után kell nézni.

A nehézséget az okozza, hogy a hálózati réteg nem rendelkezik elég információval ahhoz, hogy optimálisan megoldja a felmerülő hibát. Az optimális megoldás érdekében szorosan együtt kell működnie az alkalmazásréteggel, ami viszont sérti az OSI-modellt. Ez nem baj, sőt WSN-protokollok tervezése során egy bevált gyakorlat. Az angol szakirodalom *cross layer design*-ként hivatkozik rá [20].

Ahhoz tehát, hogy a hálózati réteg detektálhassa és lekezelhesse a hibát, együtt kell működnie az alkalmazásréteggel. Az alkalmazásréteg felé nyújtott szolgáltatásokról a prezentációs réteg gondoskodik, amit a 6.4. fejezet ír le. Ebben a rétegben lehet azokat a függvényeket elhelyezni, melyek segítségével az alkalmazásréteg szorosan együtt tud működni a most tárgyalt hálózati réteggel. Továbbolvasás előtt érdemes átfutni a 6.4. fejezetet, mert a továbbiakban az ott leírt függvények és a klaszterezés kapcsolatáról lesz szó.

Az ott biztosított függvények a klaszterezés szempontjából három csoportra oszthatóak:

- A klaszterezés szempontjából **semleges függvényeknek** tekinthetők azok, amik működését nem befolyásolja a klaszter állapota. Ebbe a kategóriába a *sendToSink(...)* és a *registerFunction* tartozik. Előbbi helyesen működik egy csonka klaszterben is, utóbbit pedig a klaszterezés előtt érdemes meghívni.
- **Igénylő függvényeknek** azok tekinthetők, amik helyes működéséhez elengedhetetlen a klaszter egészének megléte. Ebbe a kategóriába tartozik a szemafor lefoglalását és elengedését vezérlő *takeSemaphore(...)* és *releaseSemaphore()*, az őrőzkutyát vezérlő *resetWdt()*, az adatáramlást irányító **Mem(...)* függvények, a többi csomópont állapotát lekérdező *getState(...)* és a távoli függvényhívást intéző *callFunction(...)* is. Ezekben a függvényekben közös, hogy a klaszter résztvevőit próbálják elérni unicast üzenetekkel. Ha ez nem sikerül, az azt jelenti, hogy a hálózat felbomlott, az alkalmazásréteg pedig meghatározhatatlan állapotba kerül. (Nem sikerül elküldeni a tárolandó adatokat a távoli csomópontra, hibás a visszaolvasás, nem sikerült lefoglalni a közös erőforrásokat, stb...) Ez esetben az alkalmazásréteg futását meg kell szakítani, a csomópontot pedig klaszteren kívüli állapotba kell hozni.
- A harmadik csoportot a **vezérlő függvények** alkotják, amik a hálózati réteg működését közvetlenül befolyásolhatják. Ilyen az *initCommModule(...)*, ami a protokoll állapotgépeit hozza kezdeti állapotba, az *isTruncatedCluster()*, amivel az alkalmazásréteg ellenőrizheti a klaszter állapotát, valamint az *outbreak()*, mely a klaszter elhagyására (és az alkalmazásréteg megállítására) szolgál.

Az igénylő függvények specifikációja nem kíván sok magyarázatot. Ha csonka klaszterben hívódnak meg, az alkalmazásréteg nem folytatható biztonsággal, ezért inkább leáll. Ennek az a hátránya, hogy az alkalmazásrétegben nem garantálható az, hogy az ide tartozó függvények ténylegesen lefutnak. Erre figyelni kell akkor, ha a csomópontok valamilyen speciális perifériát alkalmaznak. Ilyen például az EEPROM, melyben sérülhet az adatok konzisztenciája, ha két írási művelet között újraindul az alkalmazás.

Az *initCommModule* egyetlen egyszer hívódik meg, a modul bekapcsolása után, alap helyzetbe hozza az állapotgépeket és átadja a vezérlést a klaszterépítő algoritmusnak. Az *isTruncatedCluster* függvényre azért van szükség, mert előfordulhat, hogy az alkalmazásrétegnek a leállás előtt valamit még el kell intéznie, az *outbreak()* segítségével pedig ki lehet lépni a klaszterből.

Az *outbreak()* megtehetné, hogy a hálózat többi, működőképes csomópontjának is jelzi a klaszter szétszakadását, melyek egy megfelelő üzenet hatására szintén kilépnének a hálózathoz. Ennek az lenne az eredménye, hogy az alkalmazásrétegeknek fel kellene készülni arra, hogy bármikor megállhassanak, ami egy általános protokolltól túl nagy megszorítás lenne.

Klaszteren kívüli csomópontok

Ha egy csomópont nem tud becsatlakozni egy klaszterbe, nem képes elvégezni a feladatát. Ilyenkor az alkalmazásréteg nem kapja meg a vezérlést (a megfelelő erőforrások hiányában úgysem tudna működni). Egy ilyen állapotban rekedt csomópont három dolgot tesz:

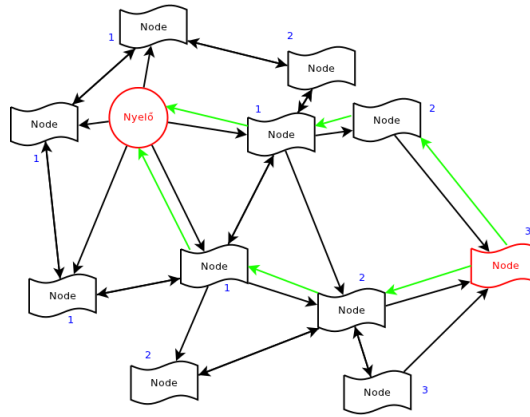
- 1.) Hallgatózik. Ha *CLUSTERBUILD*-üzenetet kap, reagál rá és megpróbál becsatlakozni egy újjonnan alakuló klaszterbe.
- 2.) Megpróbál klasztert építeni. Adott időközönként generál egy *CLUSTERBUILD*-üzenetet. Ha túl sűrűn generálja őket, feleslegesen rontja a csatornakapacitást, ha pedig túl ritkán, akkor nem jönnek időben létre a klaszterek. Ha egy csomópont klasztere felbomlik, egy rövid várakozás után érdemes szétküldenie egy *CLUSTERBUILD*-üzenetet.
- 3.) Részt vesz a klaszterezéstől független kommunikációban. A klaszterezéstől független kommunikáció felelős azért, hogy az adatok a nyelőbe megérkezzenek. Ebben minden csomópont részt vehet.

6.3.2. Klaszterezéstől független kommunikáció

Két klaszter nem tud egymással kommunikálni, viszont a nyelőknek mindenki tud üzeni. Ez Multipath Ring Routing protokollal [34] történik, melynek menete a következő:

- 1.) A csomópontok bufferelik az üzeneteket, amíg nem jön egy nyelő.
- 2.) Belép egy nyelő a hálózat területére.
- 3.) A nyelő elküld egy kérést, ami broadcast-jelleggel szétszóródik a node-ok között.
- 4.) Minden kérésnek van egy egyedi ID-je, amit a nyelő véletlenszerűen generál, valamint van egy nullától indított számlálója, amit minden, a kérést továbbító node növel (vagy konstans eggyel, vagy a vett téreőrősséggel fordítottan arányosan).
- 5.) Ha egy node olyan üzenetet vesz, aminek az ID-jét az elmúlt időszakban vette, és a számlálója (ami vagy hopot, vagy dB-t számol) nagyobb, mint az övé, akkor nem reagál rá.
- 6.) Ha egy node olyan üzenetet vesz, aminek az ID-jét még nem vette, vagy már vette ugyan, de nagyobb számlálóval, akkor elmenti magában az új számlálót és tovább küldi az üzenetet.
- 7.) Ha egy csomópont válaszolni akar egy kérésre, akkor elküld a kéréssel megegyező ID-jű válaszüzenetet, amiben jelzi a saját távolságát is a forrástól.
- 8.) A válaszüzenetet minden olyan csomópont továbbítja, ami közelebb van a nyelőhöz, mint az üzenet előző forrása.
- 9.) A válaszüzenet így robosztus módon több útvonalon jut vissza a nyelőhöz.

A fenti folyamatot a 18. ábra szemlélteti.



18. ábra. A nyelő és a forrás között kiépülő kommunikációs csatorna. A kék számok a nyelőtől mért távolságot jelzik.

6.4. Prezentációs réteg

Az alkalmazásréteget jellemzően a felhasználó készíti el, de a protokollnak biztosítania kell a megfelelő műveleteket, amelyeken keresztül az alkalmazásréteg képes kihasználni az alatta lévő rétegek funkcióit. Az alkalmazásréteg felé nyújtott szolgáltatásokról már volt szó az 5.3.3. fejezetben, most az ott is elhangzó funkciók kerülnek részletezésre. Ebben a fejezetben egy C-nyelvű API megtervezése is megtörténik, aminek a pontos megvalósítása itt még nem kerül bemutatásra.

6.4.1. Inicializáció

A tényleges futás megkezdése előtt az egyes rétegeket inicializálni kell. Be kell állítani a klaszterek méretét és az alkalmazás által igényelt memóriát. Ez egyetlen függvényhívást igényel:

```
void initCommModule(int clusterSize, int memory),
```

ahol *clusterSize* a klaszterek létszáma, *memory* pedig az alkalmazás maximális memória-igénye. Ebben nincs benne az a memória, amit párhuzamosítás során más csomópontoktól igényel. (Sem az, amit a többi réteg igényel.)

Ezt a függvényt az alkalmazásréteg nem képes közvetlenül hívni. Erről további részletek a 6.5. fejezetben találhatók.

6.4.2. A párhuzamos szakasz elérése és elhagyása

Ha egy csomópontnak szüksége van a többiek erőforrására, akkor ezt valahogy jeleznie kell feljűk. Nem szabad megengedni, hogy egyszerre két csomópont használja a klaszter összes erőforrását, mert az úgysem fog sikerülni. Erre a problémára a párhuzamos programozási paradigmák közül a szemafor nyújt megoldást. A szemafor megakadályozza, hogy egyszerre két szál (ebben az esetben két modul) akarjon hozzáférni a közös erőforráshoz. A szemafor kezeléséhez két függvényt kell biztosítani, az egyikkel azt kell jelezni, hogy a modul elérte a védett szakaszt, a másikkal pedig azt, hogy a program elhagyta azt:

```
void takeSemaphore(unsigned int wdtInit)  
void releaseSemaphore()
```

A *takeSemaphore(wdtInit)* megpróbálja megszerezni a többiek felesleges erőforrásait. Ha ez sikerül neki, akkor engedi továbbfutni a programot, egyébként felfüggeszti a működését. Ha sikerült neki belépnie a védett szakaszba, akkor minden, a klaszterben lévő fizikai modulhoz kioszt egy logikai címet, mely egy 1 és $n - 1$ közötti egész szám. A kiosztás minden *takeSemaphore(wdtInit)* hívásnál megtörténik és minden esetben véletlenszerű. A függvényhívó a többi modult ennek a számnak a segítségével éri el.

A *wdtInit* változó segítségével azt lehet megadni, hogy az őrzőkutya (angol nyelvű szakirodalomban watchdog) mennyit várjon a klaszter szétbontása előtt. Erről bővebben a 6.4.4. fejezetben lehet olvasni. (Ha *wdtInit* értéke 0 , akkor az őrzőkutya kikapcsol. Ez valódi alkalmazásban nem ajánlott, mert a modulok hibátlan program esetén is kerülhetnek olyan állapotba, hogy nem tudják elengedni a szemafort.)

A *releaseSemaphore()* elengedi a szemafort, lehetőséget biztosítva arra, hogy mások is végrehajtásuk a védett szakaszokat. A várakozó csomópontok közül ilyenkor véletlenszerű módon eldől, hogy kik folytathatják a programjuk végrehajtását. Fontos, hogy a régebb óta várakozó csomópontok nagyobb valószínűséggel kapják meg a vezérlést.

6.4.3. Klaszter állapotának ellenőrzése és a klaszter felbontása

Ha a klaszter egyik csomópontja meghibásodik, akkor a teljes klaszter működésképtelenné válik. Mivel az alsó rétegek nem végeznek rendszeres ellenőrzést a klaszter állapotára vonatkozóan, ezért a hibát az egyes csomópontok csak akkor érzékelik, amikor megpróbálnak hozzáférni a közös erőforrásokhoz. Ha az erőforrások elérésekor a protokoll hibát észlel, az alkalmazásréteg futását megszakítja, a csomópont állapotát pedig klaszteren kívülivé állítja. (A klaszterek felbomlásával és az esetleg felmerülő problémákkal a 6.3.1. fejezet foglalkozik bővebben.) Ez a hibakezelési mód az olyan alkalmazások esetén, melyek gyakran és rendszeresen fordulnak a közös erőforrásokhoz, megfelelő lehet. A dolgozatban bemutatott alkalmazás is ilyen, hiszen minden modul egy másodpercnyi mérés után elkezd a feldolgozást, amihez igénybe veszi a közös erőforrásokat. A klaszter felbomlása így egy másodpercen belül az összes résztvevő modul számára világossá válik.

Elképzelhetőek azonban olyan alkalmazások is, amelyek percenként, óránként, vagy előre meghatározhatatlan időnként fordulnak csak a közös erőforrásokhoz. Ez esetben a csomópontok rengeteg időt töltenek el egy csonka klaszterben ahelyett, hogy megpróbálnának létrehozni egy új klasztert. Mivel a fejezet célja egy, az alkalmazástól független, osztott algoritmusok végrehajtását támogató protokoll, ezért biztosítani kell az alkalmazásréteg felé olyan függvényeket is, amik segítségével ellenőrizheti a klaszter épségét és felbonthatja a klasztert.

Az

unsigned char isTruncatedCluster()

függvény igaz (nullától különböző) értékkel tér vissza, ha a klaszter csonka. Ha ez a függvény hibát jelez, az alkalmazásréteg nem veszíti el a vezérlést. Lehetősége van még néhány művelet elvégzésére. (Ilyen lehet például az EEPROM megfelelő adatokkal történő feltöltése.) A csomópont a

void outbreak()

függvény segítségével hagyhatja el a klasztert. A függvény meghívása után az alkalmazásréteg elveszíti a vezérlést. Az outbreak egy érdekes hatása, hogy megfelelő programozási technikával az alkalmazásréteg az új klaszterben onnan folytathatja a végrehajtást, ahon-

nan az előzőben abbahagyta. Erről bővebben a 6.5. fejezetben lesz szó.

Ez a két függvény jó példája az úgy nevezett *cross layer design*-nak, hiszen az alkalmazásréteg hatékonysági okokból képes közvetlenül vezérelni az alsóbb rétegek funkcióit.

6.4.4. Őrzőkutya újraindítása

Elképzelhető olyan eset, hogy a szemafor birtokló modul fizikailag meghibásodik (például lemerül az elem) vagy programhiba miatt végtelenciklusba kerül. Ekkor sohasem engedné el a szemafor, aminek az lenne az eredménye, hogy a klaszterben lévő többi csomópont a telep lemerüléséig várakozna a szemaforra.

Ezt elkerülendő, a várakozó modulok elindítanak egy időzítőt, aminek a kezdőértékét az *initCommand* függvénnyel lehet beállítani, és ha nullára ér, felbontja a klasztert. (Ha a klaszter egyik tagja meghibásodik, fel kell bontani.) Ezt az időzítőt a

resetWdt()

függvénnyel lehet újraindítani. A szemafor birtokló modulnak adott időnként meg kell hívnia ezt a függvényt. Az őrzőkutya véd a fizikai hibák ellen, a programozási hibák ellen viszont nem teljesen. Elképzelhető olyan ciklus, amiben a *resetWdt()* függvény minden iterációban meghívódik, mégsem fejeződik be soha. Az ilyen jellegű hibák valószínűsége csökkenthető, ha a ciklusokban nem kerül a függvény meghívásra.

6.4.5. Adat küldése távoli csomópontra

Egy párhuzamos algoritmus futtatása előtt az első feladat, hogy a szükséges adatokat megkapják a csomópontok. Az adatok átvitelére biztosítani kell az alábbi függvényeket:

void writeMem(unsigned char module, unsigned int address, char data)

*void writeMem(unsigned char module, unsigned int address, unsigned int size, void *data)*

A függvény első változata három paramétert vár: a *module* adja meg, hogy melyik modulnak kell megkapnia az adatokat. Ez egy egész, 0 és $n - 1$ közötti szám ahol n a klaszter mérete. 1 és $n - 1$ közötti érték esetén biztos, hogy egy távoli modul kapja meg a feladatot, 0 esetén pedig a programot végrehajtó modul képes a saját megosztott területét írni.

Az *address* a kiválasztott modul memóriacímét adja meg. 0 esetén az első rendelkezésre álló helyre írja. (Tehát nem valódi, hanem relatív memóriacímekről van szó.) Maximális értéke attól függ, hogy a modulok mennyi memóriát tartalmaznak és hogy ebből mennyit engedtek át megosztott használatra, a *data* pedig a tényleges adat, amit be kell írni.

A második változata is hasonlóan működik, csak 1 bájt helyett bármilyen adatot képes átvinni. A *size* itt azt adja meg, hogy hány bájtos a küldendő adatszerkezet, a **data* pedig az átküldendő adatszerkezet elejére mutat.

6.4.6. Adat olvasása távoli csomópontról

Nem elég az adatokat átadni, vissza is kell azt olvasni. Ez a

char readMem(unsigned char module, unsigned int address)

void readMem(unsigned char module, unsigned int address, unsigned int size)*

Az adatok olvasása hasonló az íráshoz. A *module* a célmodult adja meg (0 esetén saját magától olvas), az *address* a célcímet adja meg, a *size* pedig az áthozandó adatszerkezet méretét. Egy bájt áthozásához nem kell megadni. A függvények addig nem engedik tovább a programot, amíg nem jött meg a kért eredmény.

6.4.7. Távolról hívható függvény regisztrálása

Az eddigi függvények segítségével lehetséges a többi csomópontot távoli adattárolóként használni, de ez nem elég hatékony. Szükség van arra, hogy az egyes modulokat távolról lehessen vezérelni, ezzel a távoli adatokon különböző műveleteket hajthatnak végre.

Ehhez először jelezni kell az alsóbb rétegek számára, hogy melyik függvényeket lehet távolról is hívni és mindegyiket azonosítani kell. Erre szolgál az alábbi függvény:

```
unsigned int registerFunction(void(*func)(void*,void*))
```

A függvény bemenete egy függvénytűző, melynek paraméterei két típus nélküli tömb. (Az egyik használható a bemenetek, a másik pedig a kimenet tárolására.) Visszatérési értéke a regisztrált függvény ID-je, melyet a távoli függvényhívásnál lehet majd használni. Az ID folytonos szám, mely 0-tól kezdődik, tehát felesleges tárolni.

6.4.8. Távoli függvényhívás

A beregisztrált függvények meghívása az alábbi függvénnyel lehetséges:

```
void callFunction(unsigned int ID, int reg1, int reg2, char suspend)
```

Az *ID* a függvény azonosítója. Ez nem a párhuzamos számítást vezérlő csomóponton azonosítja a függvényt, hanem a távolin. Ezért fontos, hogy egy klaszterben minden csomópontnak ugyanazok legyenek a regisztrált függvényei.

A *reg1* és a *reg2* az írásnál és olvasásnál is látott relatív memóriacímek. A *reg1* a függvény első paramétere lesz, a *reg2* pedig a második. (Persze előbb átfordítódnak valódi memóriacímekre a távoli modulon.)

A *suspend* egy logikai érték. Megadja, hogy a program továbbfusson-e vagy megvárja az utasítás távoli végrehajtását. (Igaz esetén vár.)

6.4.9. Távoli csomópontok állapotának lekérdezése

Egy távoli utasítás végrehajtás esetén tudni kell, hogy a távoli csomópont dolgozik-e még, vagy sem. Ezt az alábbi függvénnyel lehet lekérdezni:

```
unsigned char getState(unsigned char node)
```

A függvény lehetséges visszatérési értékei:

- 0 : A modul még a legutóbb kiadott függvényt hajtja végre. Nem szabad neki semmilyen utasítást adni.
- 1 : A modul arra vár, hogy belépessen a védett zónába, ezért a rajta futó alkalmazás fel van függesztve. A számításigényes dolgokat ilyen modulokra érdemes kiosztani.

- 2 : A modul a saját programját hajtja végre. Ha kap egy távoli függvényt, akkor csak részben tud vele foglalkozni.

6.4.10. Adat küldése a nyelőnek

Ha az alkalmazás valamilyen adatot el akar küldeni a nyelőnek, akkor ezt jelezheti az alsóbb rétegeknek a

```
void sendToSink(int size, void *data)
```

függvénnyel. A *data* jelzi, hogy hol kezdődik a küldendő adatszerkezet, a *size* pedig azt, hogy ez mekkora. A Multipath Ring Routing protokoll miatt az üzenet nem küldődik el azonnal, csak egy bufferbe kerül. A tényleges küldést a nyelőnek kell kezdeményeznie.

6.5. Alkalmazásréteg

Az alkalmazásréteg feladata, hogy az alsóbb rétegek szolgáltatásaira támaszkodva elvégezzen egy hasznos feladatot. Ez a fejezet nem mutat be konkrét alkalmazásréteget, csak megadja a protokoll által támogatott felépítést és bemutat néhány egyszerű példát.

Ahogy arról már volt szó az előző réteg esetén is, a protokoll megvalósítása C nyelven történik, mivel a fordítója elérhető a legtöbb mikrovezérlőre, hatékonyan el lehet érni a hardvert, ráadásul egy széles körben elterjedt nyelv. Ezek a tulajdonságai ideálissá teszik a protokoll elkészítésére. C nyelven sajnos nincs lehetőség objektumok és névterek használatára, lehetséges azonban fordítási egységeket létrehozni. Egy lehetséges tervezési módszer, hogy az egyes rétegek külön fordítási egységben helyezkednek el. Így az alkalmazásréteg egy *application.c* nevű fájlba kerülhet. (C esetén a forrásállományok – kevés kivételtől eltekintve – a fordítási egységek.)

Ennek a fájlnak két függvényt kötelező tartalmaznia, a *void init()* függvény pontosan egyszer fut le a hálózati rétegek inicializálása előtt, a *void process()* pedig akkor, ha sikerült csatlakozni egy klaszterhez. (Tehát ez utóbbi függvény többször is lefuthat.)

Tartalmaz ezenkívül két makróhívást is, a *SET_CLUSTER_SIZE(size)* makró segítségével a klaszter méretét lehet beállítani, a *SET_APPLICATION_MEMORY(mem)* pedig az alkalmazásréteg által igényelt memóriát adja meg. (Amiből ki lehet számítani, hogy egy klaszter mennyi memóriát képes biztosítani közös használatra.) Az *initCommModule* paraméterezése ezek alapján történik meg. (A függvény tényleges meghívása automatikusan, az *init* lefutása után történik.)

Egy nagyon egyszerű *application.c* fájl tartalma a következő:

```
#include "application.h"      // A makrókat és az elérhető függvényeket tartalmazza

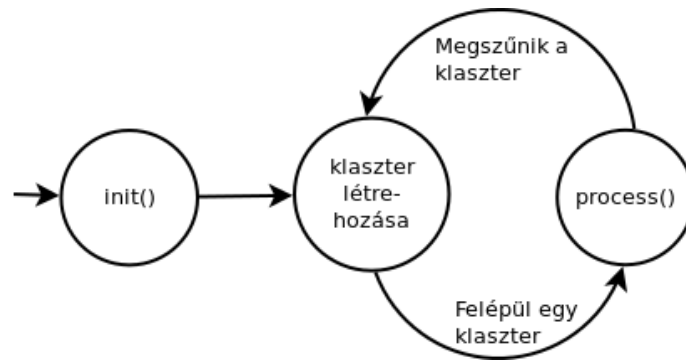
SET_CLUSTER_SIZE(10);        // A klaszterek méretét 10-re állítja
SET_APPLICATION_MEMORY(100); // Az alkalmazásréteg 100 bájt memóriát igényel

void init(){
    // Inicializációért felelős kódrészletek
}

void process(){
    // A klaszter megalakulása után ez a függvény kapja meg a vezérlést
```

}

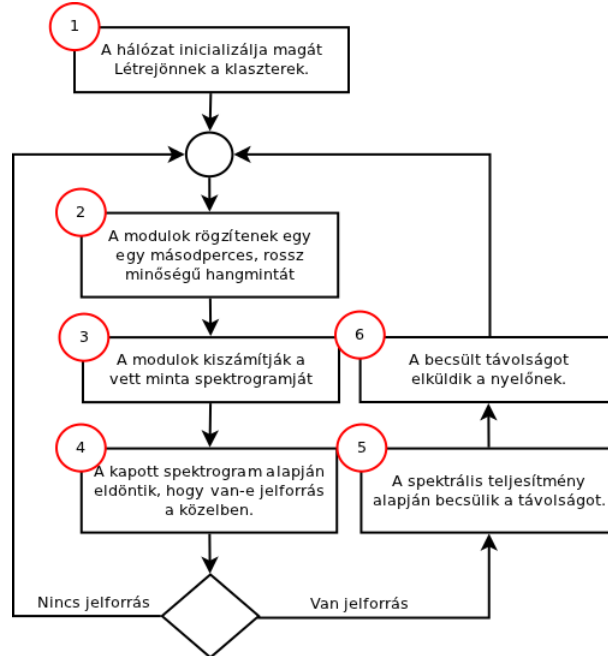
Az egyes függvények egymás utáni sorrendjét a 19. ábrán látható állapotgép szemlélteti.



19. ábra. Az alkalmazásrétegnek megfelelő egyszerűsített állapotgép.

7. Mérési és szimulációs eredmények

Ebben a fejezetben a dolgozat eredményeit és következtetéseit alátámasztó szimulációk, mérések és a belőlük levonható következtetések kaptak helyet, melyek lefedik a jeldetek-tálás teljes folyamatát (20. ábra).



20. ábra. Az ábrán a jel detektálásához szükséges lépések láthatóak

- 1.) A hálózat inicializációja során meg kell alkotni a klasztereket. Az erre vonatkozó szimulációk a 7.3. fejezetben találhatók.
- 2.) A minta detektálásához egy 1 másodperces hangmintára van szükség. Mivel a csomópontokban a memória és a számítási kapacitás korlátos, a hangmintát a lehető legalacsonyabb kvantálási szinttel és mintavételi frekvenciával kell felvenni. A 7.2.4. fejezet a mintavételi frekvencia kereséséről szól, a 7.2.5. fejezet pedig a minimális kvantálási szintet határozza meg.
- 3.) Nagyon kevés olyan alkalmazás van, melyben a hangfeldolgozást az időtartományban végzik el. Általában át kell transzformálni a jelet frekvenciatartományba a Diszkrét Fourier Transzformáció segítségével, vagy annak valamilyen változatával. Ebben a konkrét alkalmazásban a DFT nem elég (7.2.1. fejezet), de a STFT már az. Az erre vonatkozó mérések a 7.2.2. fejezetben találhatók.
- 4.) Az STFT kimenete a spektrogram, melyről már megállapítható, hogy a vett hangmintában megtalálható volt-e a jel. Ezt a 7.2.3. fejezet mutatja be.
- 5.) Ha a jelfeldolgozás eredményeként van jelforrás a közelben, akkor annak meg kell határozni a távolságát. Ezt a megfelelő frekvenciák spektrális teljesítményének a mérésével meg lehet becsülni. Az erre vonatkozó mérések a 7.1. fejezetben találhatók.
- 6.) Végül el kell juttatni a mért információkat a nyelőnek. Erre a hálózat a Multipath Ring Routing protokollt használja. Ezzel kapcsolatban nincsenek mérések, mert ez egy közismert és működő protokoll [34].

7.1. Hangforrás helyzetének bemérése

Ez a fejezet bemutat két mérést, melyek azt igazolják, hogy a spektrális teljesítmény logaritmusból jól lehet következtetni a hangforrás távolságára, egyúttal bizonyítják azt is, hogy egy hang alapú helyzetmeghatározó rendszert nem szabad egyetlen frekvenciára alapozni.

A mérések körülményei

A mérés során egy 420Hz-es hangforrás közeledett és távolodott egy Sony VPCEB2Z1E típusú laptop mikrofonjától. A távolság pár centiméter és 3 méter között változott.

Az eredmények feldolgozása

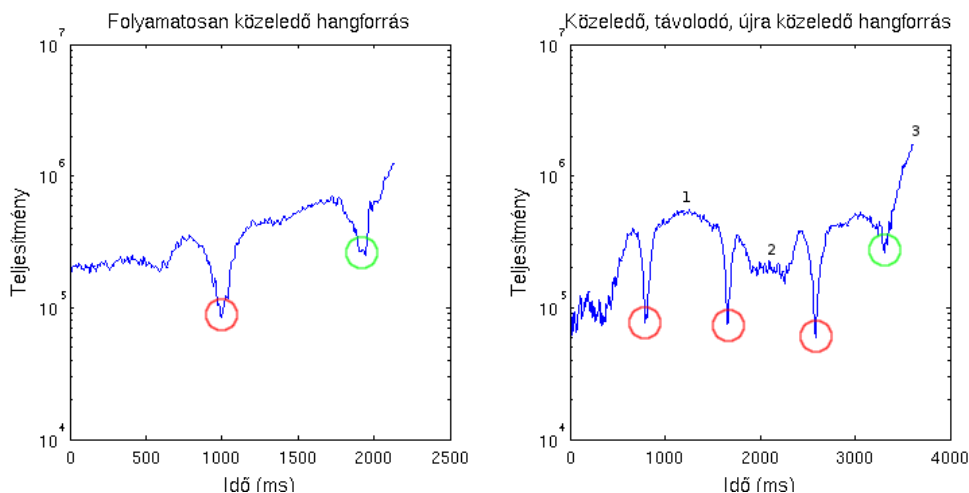
A kapott eredményt egy Matlab-függvény dolgozta fel. A függvény bemenete a feldolgozandó .wav kiterjesztésű fájl neve, kimenete pedig egy vektor, melynek minden értéke arányos az indexhez tartozó időpillanatban a hangforrás távolságával.

A Matlab-függvény működése:

- 1.) Újramintavételezi a hangmintát 2kHz-en. Erre a futási idő csökkentése miatt van szükség.
- 2.) Kiszámolja a hangminta spektrogramját. (Ehhez beépített Matlab-függvényt használ.)
- 3.) A spektrogram soraiból (amik egy adott időszakban a hangminta Fourier-transzformáltjai) kivágja a 400Hz és 500Hz közötti szakaszt.
- 4.) Elkészíti a kimenetet. A kimenet minden eleme egy sor (vagyis a 400Hz és 500Hz közötti tartomány) összegzése. Az így kapott kimenet megadja, hogy hogyan függ az időtől a 400Hz és 500Hz közötti spektrum teljesítménye.

A mérési eredmények

A 21. grafikonokon láthatóak a mérési eredmények.



21. ábra. A fenti ábrán két mérés látható. Mindkettő egy 420Hz-es hangforrás vett teljesítményének logaritmusát mutatja az idő függvényében, miközben a forrás közelített és távolodott a mikrofontól. Mindkét esetben néhány centiméter és három méter között változott a távolság.

A bal oldali grafikonon az látszik, hogy a forrás három méterről néhány centiméterre megközelítette a nyelőt közel egyenletes sebességgel. A függvény két tuskét leszámítva monoton növekvő. (Az egyik piros, a másik zöld körrel van jelölve.) Ezeket a tuskéket a többutas terjedés miatt bekövetkező fading okozza. Ez a jelenség a vezeték nélküli kommunikációban is nagy problémákat okoz és általában FHSS-sel lehet ellene védekezni. Ha fix 420Hz-es hang helyett a jelforrás változtatja a frekvenciáját, a fading teljesen kivédhető.

A jobb oldali grafikonon a forrás 3 méterről 1200ms alatt megközelíti a mikrofont 1.5 méterre (1.), majd eltávolodik 2 méterre (2.) végül eléri a mikrofont (3.). A hangforrás összesen háromszor halad át a pirossal jelölt fading-en, egyszer pedig a zölddel jelöltön. A fading-eket leszámítva (ami szélesebb spektrumú detektálható jelet használva kiküszöbölhető) a mikrofontól mért távolság könnyen kiszámolható a vett teljesítmény logaritmusából.

Következtetések

A mérésekből látszik, hogy a mikrofon 2.5-3 méteres sugarú körben nagy pontossággal képes meghatározni a hangforrások távolságát, de nem szabad egyetlen frekvenciára alapozni a detekciót.

7.2. Hangfeldolgozás lépései

Az előző mérésből kiderült, hogy a detektálható hang nem állhat egyetlen frekvenciakomponensből, hiszen akkor a fading miatt nem lehetne megbízhatóan meghatározni a hangforrások helyzetét. Szükség van valamilyen frekvenciamodulációra. Például egy 450Hz-es középfrekvenciájú, 100Hz sáv szélességű, 1Hz-es szinusszal frekvenciamodulált jelalakra, melynek spektrogramja a 23a. ábrán látható. Egy ilyen jelalak elég nagy spektrumra van szétszétva ahhoz, hogy a fading jelensége ne legyen számottevő, viszont nehezebb is detektálni. Ez a fejezet egy ilyen jelnek a detektálásáról szól.

7.2.1. A DFT tesztelése

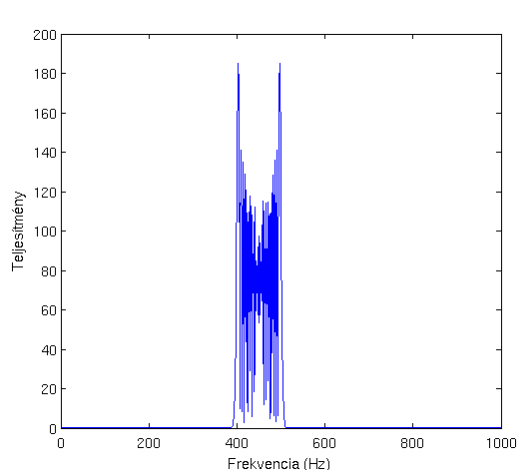
A mérés során egy frekvenciamodulált (FM) jelet kell feldolgozni DFT-vel.

A mérés körülményei

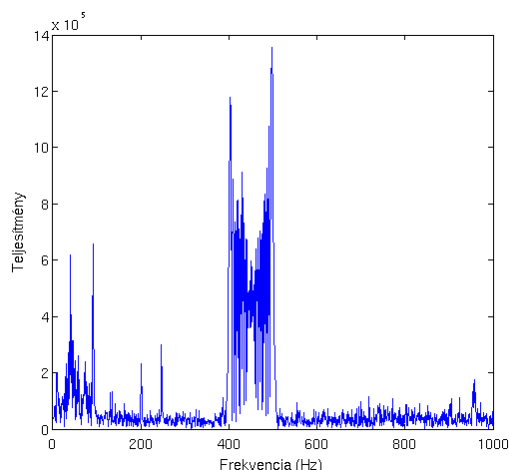
A bevezetőben is említett jelet a FuncGen nevű program generálta, mely ingyenesen hozzáférhető a Google Play áruházból. A felvétel egy Sony VPCEB2Z1E típusú laptop mikrofonjával készült, a feldolgozást pedig a Matlab beépített függvényei végezték. A jel több fajta (ideális, zajos, ...) környezetben lett felvéve.

Az eredmények feldolgoása

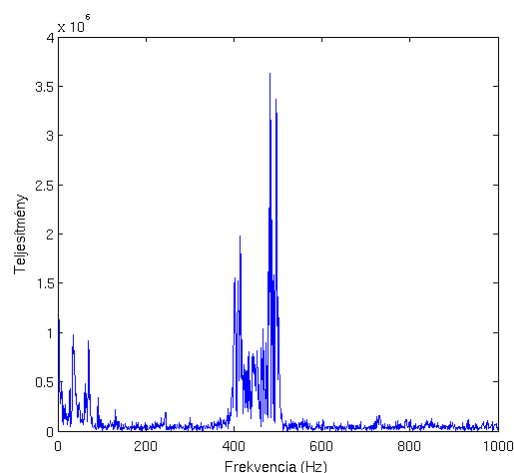
A kapott hangfelvételek Matlab program segítségével lettek feldolgozva. Első lépésként újra lettek mintavételezve 2kHz-el, hogy gyorsítani lehessen a szimulációt. Ezután a Matlab beépített fft-függvényével az összes minta át lett alakítva frekvenciatartományba. A mérések eredménye a 22. ábrán látható.



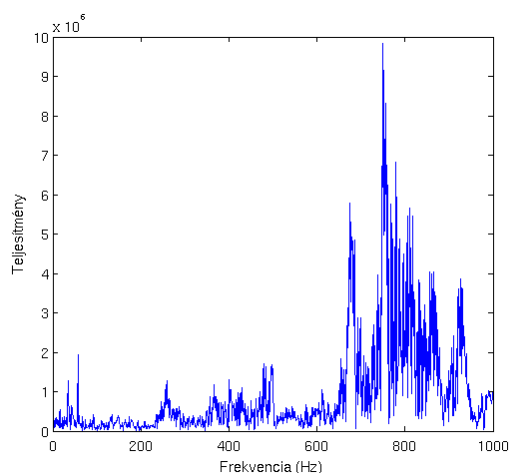
(a) Ideális felvétel



(b) Közeli felvétel



(c) Távoli felvétel



(d) Zavart felvétel

22. ábra. Különböző körülmények között felvett hangok Fourier-transzformáltja. A bal felső ábrán egy, a Matlab `fmmod`-függvényével generált felvétel képe látható. Nincs rajta semmilyen zaj, jól látszódnak a frekvenciamodulált jel jellegzetességei. A jobb felső ábrán egy valós felvétel Fourier-transzformáltja látható. A felvételen tisztán hallatszik a frekvencia-moduláció, a képen pedig (az ideális jelalakhoz képest) enyhe torzulások tapasztalhatóak. A jelforrás és a mikrofon között egy méter alatt volt a távolság. A bal alsó ábrán a jelalak jelentősen torzult. Ennek az az oka, hogy a mikrofon és a hangforrás között 3 méterre nőtt a távolság. A jelalak ettől még felismerhető. A jobb alsó ábra egy olyan hangfelvételtől készült, ahol a detektálandó jelen kívül jelen van egy másik, szabálytalan frekvenciamodulált jel is. Bár a felvételen hallható az eredeti jel, a spektrum képe erősen torzult.

Következtetések

Az eredményekből látszik, hogy a DFT nem tűri jól a zajt. Ennek az az oka, hogy az egyes frekvenciák teljesítményét egy viszonylag hosszú időablakra (a teljes felvételre) átlagolja. Egy olyan elemzési módszerre van szükség, amivel érzékelteni lehet a frekvenciák időbeli változását.

7.2.2. A STFT eredményének feldolgozása

Ez a mérés azt bizonyítja, hogy az STFT-vel azok a minták is feldolgozhatóak, amik DFT-vel nem, ráadásul a feldolgozás sokkal egyszerűbb.

A mérés körülményei

A mérések során a 7.2.1. mérésben is bemutatott hangminták elemzése történik meg STFT segítségével.

Az eredmények feldolgozása

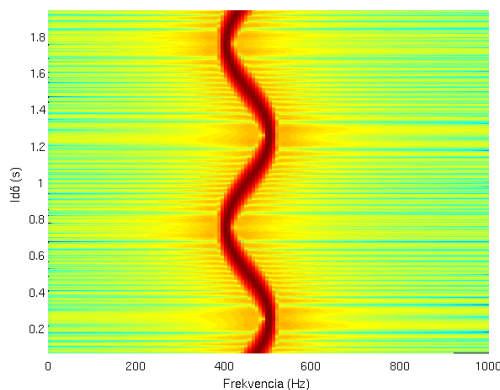
Az itt bemutatott képeket a matlab spektrogram-függvénye generálta. A felhasznált paraméterek:

$$\text{spectrogram}(y, 256, 250, 256, F_s);$$

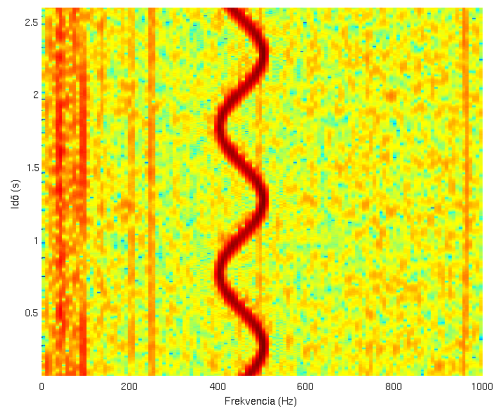
ahol y egy 2000Hz-en mintavételezett hangminta, F_s pedig a mintavételi frekvencia (2000).

A mérési eredmények

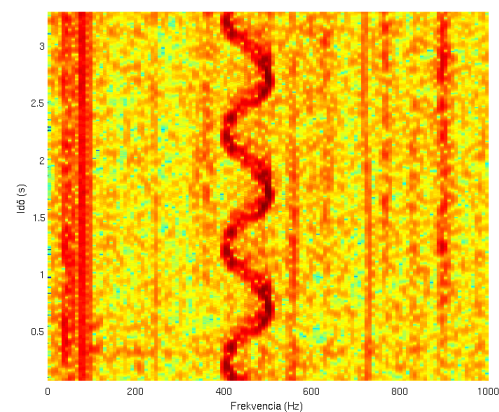
A mérés eredményei a 23. ábrán láthatóak.



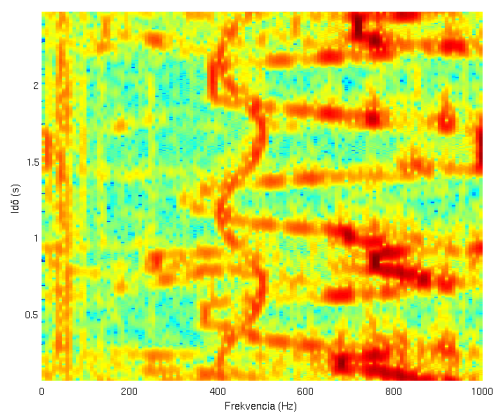
(a) Ideális felvétel



(b) Közeli felvétel



(c) Távoli felvétel



(d) Zavart felvétel

23. ábra. Különböző körülmények között felvett hangok spektrogramja.

A bal felső ábrán egy, a Matlab `fmmod`-függvényével generált felvétel képe látható. Nincs rajta semmilyen zaj, látszanak a frekvenciamodulált jel jellegzetességei.

A jobb felső ábrán egy valós felvétel spektrogramja látható. A felvételen tisztán hallatszik a frekvencia-moduláció, a képen pedig (az ideális jelalakhoz képest) halvány zajok láthatóak. A bal alsó ábrán a jelalak enyhén torzult. Ennek az az oka, hogy a mikrofon és a hangforrás között 3 méterre nőtt a távolság. Az ábrán tapasztalható elmosódást a többutas terjedés okozza.

A jobb alsó ábra egy olyan hangfelvételtől készült, ahol a detektálandó jelen kívül jelen van egy másik, szabálytalan frekvenciamodulált jel is. A felvételen jól kivehető mindkét jel. Érdekes összevetni ezt az ábrát a 22d. ábrával. A két kép ugyan arról a hangmintáról készült, csak más eljárással. Látszik, hogy az STFT képes elválasztani az időben elkülönülő, azonos frekvenciájú komponenseket is.

Következtetések

A kapott spektrogramokon minden jelalak könnyen kivehető és azonosítható. A hangminták elemezhetőek vele.

7.2.3. STFT eredményének kiértékelése

Az itt bemutatott mérések segítségével az 5.2.2. fejezetben bemutatott képelemző algoritmus helyességéről lehet meggyőződni. Az algoritmus itt nem kerül ismertetésre.

A mérés körülményei

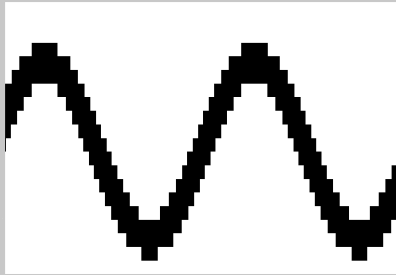
Az itt elemzett hangminták a 7.2.1. és a 7.2.2. mérésben is bemutatásra kerültek.

Az eredmények feldolgozása

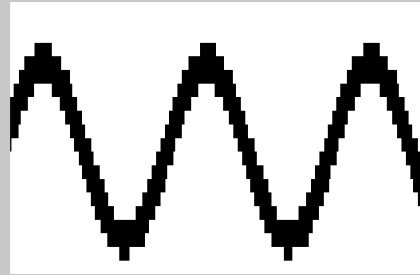
Az eredményeket egy egyszerű Matlab-függvény elemezte ki. A függvény bemenete a spektrogramot tartalmazó mátrix, kimenete pedig a bináris kép, valamint a fekete és fehér pixelek százalékos aránya.

A mérési eredmények

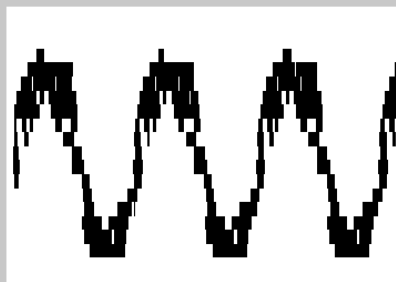
Az eredmények a 24. ábrán láthatóak.



(a) Ideális felvétel - 17%



(b) Közeli felvétel - 16%



(c) Távoli felvétel - 18%



(d) Zavart felvétel - 22%



(e) Halk beszéd - 8%



(f) Hangos zörej - 43%

24. ábra. Különböző körülmények között felvett hangok elemzésének eredménye. A felirat utáni százalék azt adja meg, hogy a kép hány százaléka fekete.

A bal felső ábrán egy, a Matlab `fmmod`-függvényével generált felvétel képe látható. Nincs rajta semmilyen zaj, látszanak a frekvenciamodulált jel jellegzetességei. Összehasonlítva a 23a. ábrával, a különbség nem számottevő. Ugyanez elmondható a jobb felső ábráról is. A (c) ábrán már látni lehet egy kis torzulást, de ez se számottevő.

A (d) ábra, ami a zavart felvétel alapján készült, szintén könnyen felismerhető.

Az (e) és (f) ábrák csak referenciaként szolgálnak. Egyik hangfelvétel sem tartalmazza az eredeti jelet, ennek megfelelően a kapott arány is szélsőséges (50% közeli és 10% alatti) lett.

Következtetések

A 24. ábra képeiről látszik, hogy ha a megfelelő jel benne van a hangmintában, akkor az algoritmus kimenete 15% és 25% közötti. Az itt bemutatott mérések alapján az algoritmus - bár több empirikus paraméter is szerepel benne - megfelelő teljesítményt ad.

7.2.4. Alulmintavételezés

Ez a mérés azt mutatja be, hogy egy 450Hz-es vivőfrekvenciára ráültetett, 1Hz-es frekvenciamodulált szinusz-jelet (melynek a teljes sávszélessége 100Hz), nem kell 1000Hz-en mintavételezni. Mivel a memória nagyon korlátozott erőforrás egy WSN esetén, ezért a mintavételi frekvenciát érdemes minél alacsonyabban tartani.

A mérés körülményei

A mérés során felhasznált minta nem valós felvétel, hanem a matlab `fmod`-függvénye generálta. A függvény az alábbi paraméterezéssel lett meghívva:

$$\text{fmod}(x, 450, 1000, 50);$$

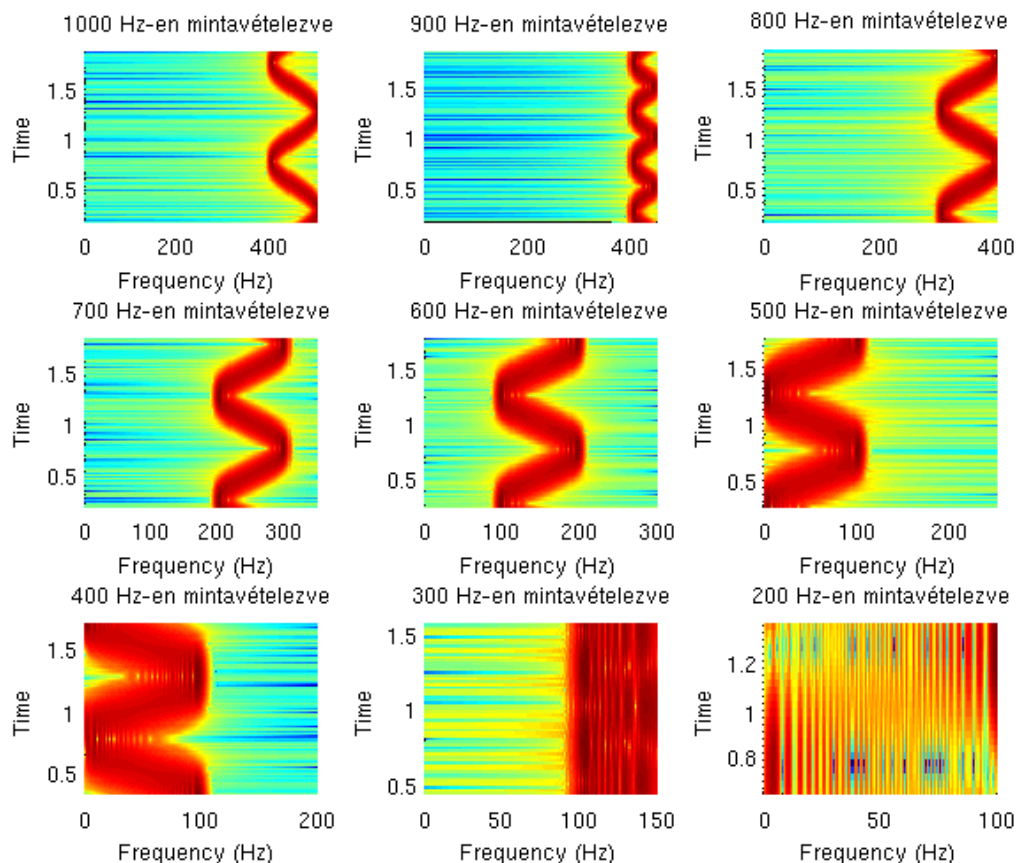
ahol x egy 1Hz-es szinuszjel, 450 a vivőhullám frekvenciája, 1000 a mintavételi frekvencia és 50 pedig a frekvenciaeltérési állandó.

Az eredmények feldolgozása

A mérés során az eredetileg 1kHz-en mintavételezett jel lett újamintavételezve 900Hz-en, 800Hz-en, ..., 200Hz-en. A kapott jelalakoknak minden lépésben ki lett számolva a spektrogramja, melyek az eredmények között láthatóak.

A mérési eredmények

A mérés eredménye a 25. ábrán látható. A mintavételezési frekvencia csökkenésével a 100Hz sávszélességű jel ugyan átlapolódik, de felismerhető marad.



25. ábra. Az egyes képeken ugyan annak a jelnek a spektrogramja látható egyre kisebb mintavételezési frekvenciával.

Következtetések

A jel 400Hz-es mintavételi frekvencia mellett még felismerhető. 300Hz-cel mintavételezve már elveszíti minden jellegzetességét. Az egyetlen különbség a képek között, hogy a függvény képe fokozatosan elkenődik.

7.2.5. Az algoritmus hatékonysága alacsony mintavételi frekvenciákon, egy bites kvantálással

A mérés feladata azt kideríteni, hogy mi az a minimális mintavételi frekvencia és kvantálási szint, ami mellett a jeldetektáló algoritmus még elfogadható eredményt ad.

A mérés körülményei

A mérés során a 7.2.3. fejezetben is bemutatott hangminták lesznek újra kielemezve, de ezúttal egybites kvantálással és alacsony mintavételi frekvencián. A mért és újrakvantált hangmintákat az 5.2.2. fejezetben bemutatott algoritmus dolgozza fel. Az algoritmusban egyedül a threshold értéke változott $\frac{E_{max}}{3.2}$ -ra.

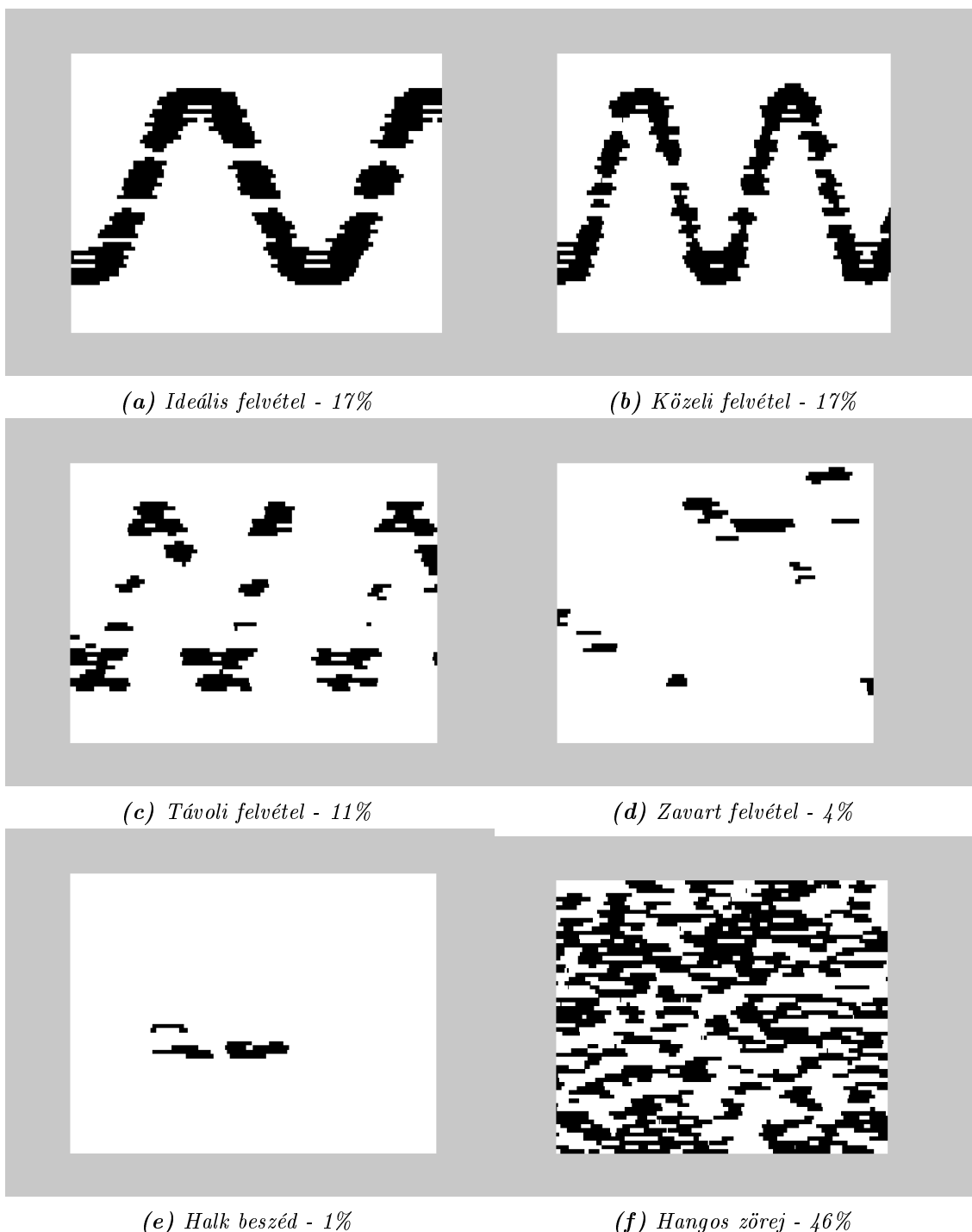
A mérési eredmények

A mérési eredmények a 26. ábrán láthatóak.

A 26b. ábrán egy valós felvétel spektrogramja látható. A felvételen tisztán hallatszik a frekvencia-moduláció, a képen pedig (az ideális jelalakhoz képest) enyhe különbségek láthatóak.

A 26c. ábrán a jelalak kivehető, de csak a szélső pontokban.

A 26e. és a 26f ábrák referenciák. Egyik egy halk beszéd töredékének a felvétele, a másik pedig hangos kaparászásé. Egyikben sincs benne a keresendő jel.



26. ábra. Különböző körülmények között felvett hangok elemzésének eredménye. A hangminták megegyeznek az 5.2.2. fejezetben szereplőkkel, de itt egybites kvantáló kvantálta a jeleket, a mintavételi frekvencia pedig 600Hz.

Az (a) ábrán egy, a Matlab `fmmod`-függvényével generált felvétel képe látható. Bár vannak rajta szakadások, a jel jól kivehető.

A (b) ábrán egy valós felvétel spektrogramja látható. A felvételen tisztán hallatszik a frekvencia-moduláció, a képen pedig (az ideális jelalakhoz képest) enyhe különbségek láthatóak. A felvétel zajszegény környezetben, a mikrofonhoz közel készült. A (c) kép ennél sokkal kevésbé felismerhető, de a szinuszhullámok szélsőértékeit ki lehet venni. A (d) képen nem látni semmit. Ez a kép a zavart jelhez tartozott. Bár ezt az algoritmus nem képes érzékelni, ez elfogadható hiba, hiszen katasztrófa sújtotta területeken nincs olyan hangforrás, ami szélesspektrumú zavaróként működhetne.

Következtetések

Az alacsony mintavétel miatt a magasabb (300Hz feletti) frekvenciák is a $(0Hz, 300Hz)$ közötti tartományba képződik le, ahol az zajként jelentkezik. Emiatt a 26c. ábrán lévő jelalak halvány lesz, de még is így benne vannak a szinusz jellegzetes hurkai és a százalék is 10% és 25% közötti, így még elfogadható.

A 26d. ábrán látható mintán ugyan rajta volt az eredeti jel, de az algoritmus már nem volt képes detektálni. Ennek az az oka, hogy ebben a mintában széles sávú zavarás volt jelen, ami teljes egészében a $(0Hz, 300Hz)$ tartományba lapolódott, így már képtelenség volt detektálni.

Végző következtetésként az algoritmus működik 600Hz-es mintavétellel is, a nem túl szélsőséges eseteket helyesen kategorizálja be.

7.3. Protokoll hatékonyságának tesztelése

Ebben a fejezetben azok a szimulációs eredmények kaptak helyet, melyek a klaszterezés hatékonyságát támasztják alá. A szimulációk Castalia [35] segítségével készültek. A Castalia elsősorban WSN és PAN tesztelésére lett kifejlesztve. Azért a Castalia-ra esett a választás, mert ez egy kipróbált és sokak által használt környezet.

A klaszterezés hatékonyságát két érték határozza meg: a létrejött klaszterek száma és a klasztereken belüli átlagos térerő.

A továbbiakban három különböző topológia kerül bemutatásra, ahol a csomópontok 3 tagból álló klasztereket hoznak létre.

7.3.1. Klaszterezés hatékonysága

A mérés során a modulok különböző topológiákban hoznak létre háromtagú és öttagú klasztereket. A mérések során változik a csomópontok száma (x tengely) és sűrűsége (r).

A mérés körülményei

A szimuláció a már említett Castalia környezetben lett elkészítve. Összesen három különböző szimuláció készült:

- Először a modulok egymás mellett sorban helyezkedtek el. Minden modulnak két közvetlen szomszédja volt, ezért itt csak háromtagú klasztereket kellett létrehozni.
- Majd a modulok egy négyzetrács pontjaiban helyezkedtek el. Minden modulnak négy szomszédja volt. Itt öt tagú klasztereket kellett létrehozni.
- Végül a modulok egyenletes eloszlás szerint lettek elhelyezve egy meghatározott területű négyzeten. Ez hasonlít legjobban a valós körülményekhez. Itt csak öttagú klasztereket kellett létrehozni.

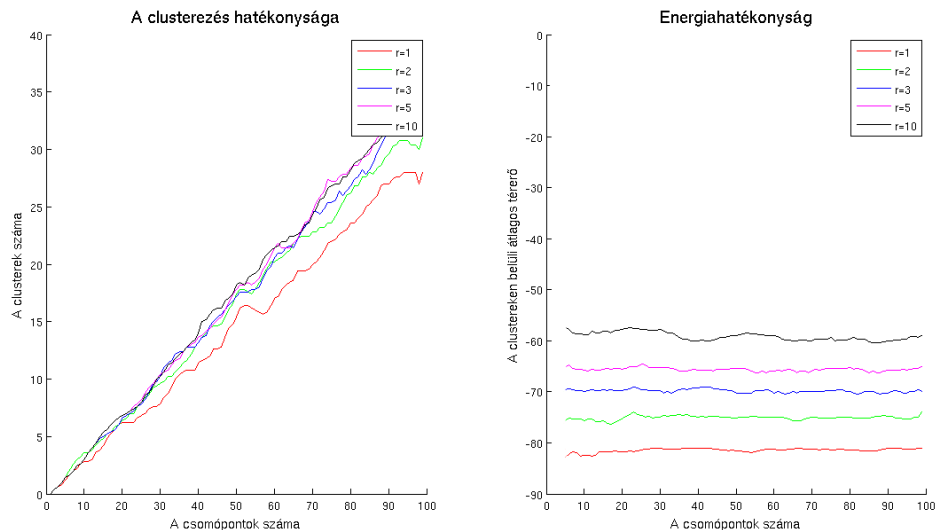
Az eredmények feldolgozása

A szimuláció kimenete néhány logfájl volt, amiből Matlab segítségével meg lehetett állapítani, hogy hány klaszter jött létre és hogy a klasztereken belül mekkora az átlagos térerősség.

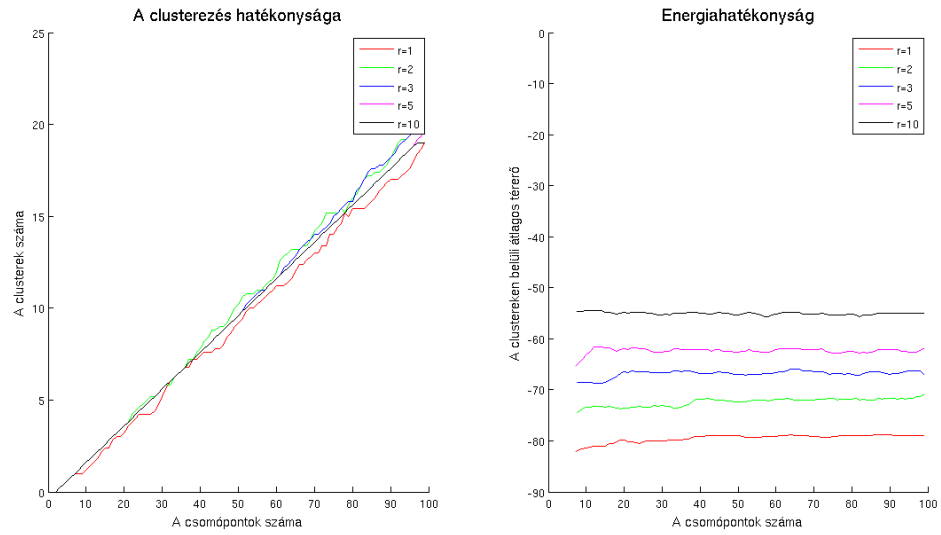
A mérési eredmények

A 27. ábrán látható a sortopológiához tartozó klaszterezés. Itt csak háromtagú klaszterek létrehozása történik. A csomópontok száma lineáris növekedést mutat. Azért látható benne zaj, mert a csomópontok véletlenszerű időt várnak a klaszterezés előtt. Az Energiahatékonyság (bal oldali grafikon) láthatóan $r = 1$ -nél a legkisebb, hiszen ilyenkor vannak egymástól legtávolabb a node-ok. (Az $r = 1$ jelentése, hogy a node-ok mindkét irányban csak egy másik node-ot látnak. $r = 10$ esetén 10 node-ot látnak mindkét irányba.) A grafikonról leolvasható, hogy a csomópontok száma nem befolyásolja az átlagos térerőt a klasztereken belül.

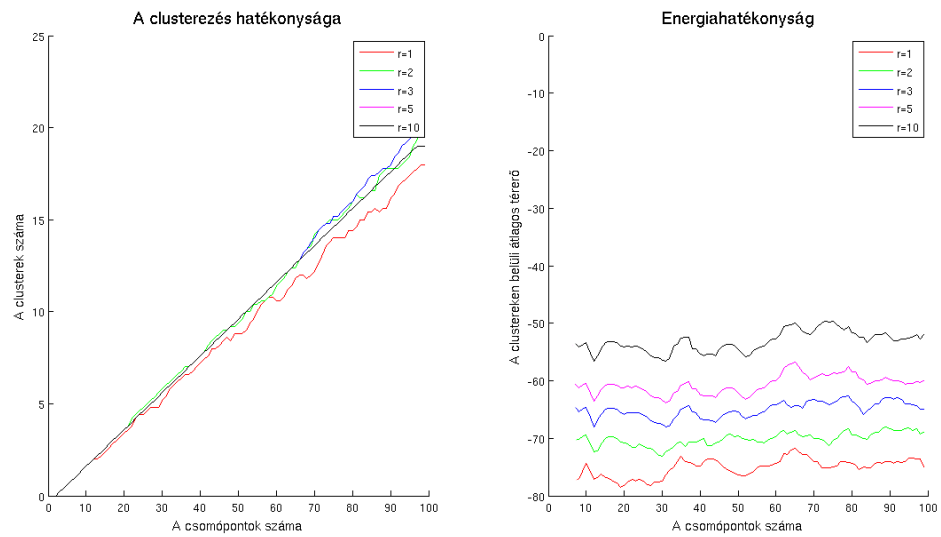
A 28. ábrán négyzetrácsos (vagyis grid) topológiában vannak rendezve a csomópontok és 5 tagú klasztert kell létrehozniuk. Végül a 29. ábrán lévő szimuláció azt mutatja meg, hogy a protokoll mennyire hatékony, ha a csomópontok egyenletes valószínűséggel, de véletlenszerűen vannak lerakva. Ez utóbbi közelíti a legjobban a valóságot. A grafikonok alapján elmondható, hogy az algoritmus megfelelő hatékonysággal állítja össze a klasztereket. Az energiahatékonyság viszont magyarázatra szorul. Látszólag összefüggés van a csomópontok száma és az átlagos térerősség között, hiszen a különböző sűrűségeknek megfelelő görbék hasonlítanak egymásra. Ez csak a szimuláció módja miatt van így: A szimuláció során először le lett rakva egy megadott számú csomópont véletlenszerűen, aztán ebben az elrendezésben mérte meg az átlagos térerősséget akkor, ha a két node közötti átlagos távolság megegyezik az adótávolsággal ($r = 1$), akkor ha megegyezik az adótávolság felével ($r = 2$) és így tovább.



27. ábra. 3 tagú klaszterek létrehozása lánc-topológia esetén



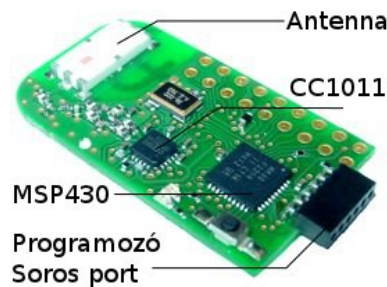
28. ábra. 5 tagú klaszterek létrehozása grid-topológia esetén



29. ábra. 5 tagú klaszterek létrehozása, egyenletes eloszlás

8. Implementáció

A fejezet célja annak igazolása, hogy a dolgozat során megtervezett protokoll implementálható és valós környezetben is jól teljesít. A megvalósítás során AMB8423 típusú modulok kerültek felhasználásra, melyek több szempont miatt is ideálisak egy szenzorhálózat gyors kifejlesztésére. A modulok megvalósításán kívül a fejezet bemutatja a mobiltelefonon futó alkalmazást is, de a nyelőről nem lesz szó, bár ez is fontos része a rendszernek. A nyelő az az eszköz, ami a mentőegységek felé prezentálja az adatokat. Fontos, hogy a mentőegységek egy megfelelően feldolgozott, jól átlátható jelentést kapjanak a túlélők valószínűsíthető helyzetéről.



30. ábra. Az AMB8423-as modul és főbb részei

8.1. A node-ok implementálása

A csomópontok fizikai megvalósítása AMB8423-as modulokon [36] történt, melyeken egy CC1011 RF-chip és egy MSP430F2274 mikrovezérlő működik. Ezek a modulok a beépített, nagy kapacitású mikrovezérlő, kis méretük, egyszerű programozhatóságuk, beépített státuszjelző LED-jeik és könnyen elérhető soros portjuk miatt ideálisak egy protokollok implementálására és teljesítőképességük analízisére.

8.1.1. A fejlesztőkörnyezet bemutatása

A modulokban található MSP430-architektúrájú chip nagyon egyszerűen, több fejlesztő-eszköz segítségével is programozható. Először az IAR Embedded Workbench Kickstart Edition segítségével történt meg a programozás. Azért erre esett a választás, mert egyszerűen tartalmazza az égetőt, fordítót, IDE-t és a nyomkövetőt, ráadásul rengeteg tananyag és leírás érhető el róla az interneten. Rendelkezik ingyenes verzióval is, mely 4kB-ban limitálja a kódméretet. Ez elfogadható kompromisszumnak tűnt. Az ingyenes verziót az IAR weblapjáról lehet letölteni, telepítés után pedig regisztrálni kell az ingyenes próbalicence-hez.

A fejlesztőkörnyezeten belül először létre kell hozni egy munkalapot (Workspace), ezután ehhez tetszőleges számú projektet lehet hozzáadni. Minden projektben külön be lehet állítani a célchipet és az égetőt. A projektek közötti váltás egyszerű és gyors.

8.1.2. A modulok tesztelése

A fejlesztőeszköz beállítása után a modulok kipróbálása a következő feladat. A teszt-program megírása során szempont volt, hogy gyors és látványos eredményt adjon, mivel hatvan modult kellett kipróbálni. Ez a program a klasszikus, mikrovezérlős környezetben

a "Hello World"-nek megfeleltethető LED-villogtatás lett. Miután minden modul hibátlannak tűnt, el lehetett kezdeni a tényleges munkát. Ez a mikrovezérlő külső és belső perifériájának megismerésével és kipróbálásával kezdődött.

Először a modulokon található két LED-et és nyomógombot kezelő szoftver készült el. Ennek oka, hogy a legegyszerűbben ezekkel az eszközökkel lehet kommunikálni egy felhasználóval. Ezután a modulokban található soros port számítógépre kötése történt meg, így már összetettebb információkat is át lehetett vinni a modul és a felhasználó között. Végül a modulokon található rádiós modult kellett felélesztetni.

8.1.3. SimpliTI-projekt

A modulokon található rádiós chip felélesztése a Texas Instruments SimpliTI protokolljának a segítségével történt meg. A SimpliTI egy kis méretű, MSP-architektúrára kifejlesztett C-nyelvű függvénykönyvtár, ami képes a legtöbb Texas Instruments-gyártmányú rádiós chippel kommunikálni SPI-interface-en keresztül. Előnye, hogy elvégzi a modul inicializálását, kezeli a modul felől érkező megszakításokat és képes csomagokat küldeni, valamint teljes mértékben skálázható és testre szabható.

A SimpliTI alapértelmezésként maximum 29 bájtos csomagokat küld és fogad, melynek első bájtja a csomag hossza, az azutáni 8 (4-4) pedig a küldő és a fogadó címe. Ezeket a rendszer alapértelmezésként nem veszi figyelembe, így a fejlesztő teljes szabadságot kap a fizikai réteget illetően. Bár a csomag minimum méretére nincs megkötés, az implementáció során nyert tapasztalatok alapján 14 bájtnál kisebb csomagokat nem képes kezelni.

8.2. A protokoll implementálása

Az implementálás során a cél az alkalmazásréteg felé nyújtott függvények (lásd 4.1.3. alfejezet) implementálása volt. Az implementáció kezdetén az IAR fejlesztőeszköz nem engedett 4kB-nál nagyobb programot írni, ezért csak a klaszterezés fért el a kódban, a klasztereken belüli kommunikáció már nem. Folyamatban van a projekt portolása mspgcc nevű fordító alá, ami teljesen ingyenes, korlátozásmentes és több nagy cég (pl. 77elektronika) is használja. Ebben a fejezetben a klaszterezés implementálása kerül bemutatásra. Az implementálás a már ismerttetett OSI-modell alapján történt.

8.2.1. Fizikai réteg

A fizikai réteg feladata a csomagok fogadása, küldése és a címzés kezelése. Az ide tartozó függvények a *phisLayer.c* nevű fordítási egységben kaptak helyet, a többi réteg felé nyújtott szolgáltatásokat (függvényeket) pedig a *phisLayer.h* fejléclomány tartalmazza.

A *phisLayer.c* tartalmaz egy 4 bájtos fizikai címet, melyet bármelyik réteg lekérdezhet a *getAddress* függvény segítségével, az első két bájtát pedig módosíthatja a *setAddress* segítségével. A második két bájt programozáskor kerül beégetésre, futás közben nem lehet módosítani és garantáltan egyedi. Ezzel a címzési mechanizmussal meg lehet valósítani a már említett háromlépcsős címzést, hiszen a fizikai réteg három esetben adja tovább a felső rétegeknek a fogadott csomagot:

- Ha az a 0:0:0:0 broadcast-címre érkezett, ami csupa nullát tartalmaz.

- Ha az az x:y:0:0 címre érkezett, ahol x:y megegyezik a modul címének első két bájtjával
- Ha az az x:y:z:a címre érkezett, ahol x:y:z:a a modul tényleges címe.

A réteg megvalósít egy *sendSegment* függvényt, ami bemenetnek vár egy címzettet és egy üzenetet, és amint lehet elküldi azt a tényleges címzetteknek. A fizikai réteg egyedül az adatkapcsolati réteg szolgáltatásait veszi igénybe, azt is csak akkor, ha a modul kapott egy üzenetet.

8.2.2. Adatkapcsolati réteg

Az adatkapcsolati réteg a *dataLinkLayer.c* fordítási egységben helyezkedik el, a szolgáltatásait pedig a *dataLinkLayer.h* definiálja. A réteg feladata, hogy biztosítsa azt, hogy az üzenetek ne vesszenek el két node között. Ezt úgy éri el, hogy ha a felsőbb rétegektől a *sendPacket* függvényen keresztül kap egy csomagot, ami nem broadcast-címre megy, akkor a csomag elküldése után elindít egy időzítőt, ami 1ms után jelez. Ha eddig nem érkezett az üzenetre nyugta, akkor újraküldi a csomagot. Az újraküldés legfeljebb háromszor történik meg. Ha harmadikra sem kap nyugtát, akkor a hibát jelzi a felsőbb rétegeknek.

Mivel az elfogadhatatlan lenne, hogy az alkalmazásréteg futása az üzenet küldéséig (ami akár 3ms is lehet) felfüggesztődik, ezért a felsőbb rétegek felé nem tud a *sendPacket* visszatérési értékén keresztül üzeni. Ehelyett biztosítja a *getLastPacketState* függvényt, amivel a felsőbb rétegek információt kérhetnek a legutoljára küldött csomag állapotáról. A *getLastPacketState* a visszatérési értéke lehet 0, ez esetben még zajlik a küldés, lehet 1, ez esetben nem sikerült elküldeni harmadik próbálkozásra sem, vagy lehet 2, ami a sikeres küldést jelöli.

Szintén ez a réteg felelős azért, hogy a fizikai réteg felől kapott csomagokra küldjön egy visszaigazolást. Broadcast-üzenetekre nem küld és nem is vár visszaigazolást, hiszen a feladó nem tudhatja, hogy kik fogják megkapni az üzenetét.

A visszaigazolás-funkció nem fért bele a már említett 4kB-is programmérethatárba, így feltételes fordítással ki kellett szedni a kódból. A későbbi log-okban ezért nem látszik az ACK. Ettől függetlenül a funkció tesztelése megtörtént, a réteg szolgáltatásai benne vannak a rendszerben és az mspgcc-re történő portolás után újra élesítve lesznek.

8.2.3. Transzport réteg

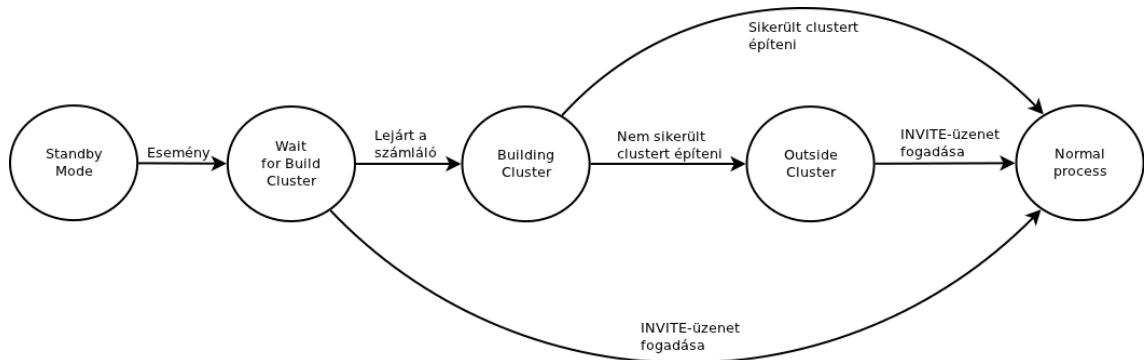
A transzport réteg feladata, hogy az alkalmazásrétegnek ne kelljen foglalkoznia azzal, hogy mi történik az alsóbb rétegekben. Logikusnak tűnt tehát ebben a rétegben megvalósítani a klaszterezéssel és kommunikációval kapcsolatos API-t. A téma szempontjából ez a legfontosabb réteg. Jelenleg két függvényt biztosít: a *getPacket* segítségével az alatta lévő rétegek adhatnak át valamilyen üzenetet, az *initCluster* segítségével pedig az alkalmazásréteg kezdheti meg a klaszterezést. Ez utóbbinak az egyetlen paramétere a klaszter mérete.

A csomópontok a működésük során hat állapotban lehetnek (31. ábra), melyek jelentése a következő:

- 1.) **Standby mode** : készenléti állapot. A node-ok valamilyen jelre várnak, hogy megkezdhesék a működésüket. A tápellátás bekapcsolása után ez a csomópontok

alapértelmezett állapota. A jel, ami ebből az állapotból kibillentő őket, jelenleg egy "GO" tartalmú broadcast-üzenet, amiről a későbbiekben lesz szó. Ebben az állapotban a modul státusz-LED-jei (egy piros és egy zöld) sötétek.

- 2.) **Wait for build cluster** : a csomópont visszaszámol egy véletlen értékről. Ebben az állapotban arra vár, hogy lejárjon a számlálója és megpróbálhasson építeni egy saját klasztert. Közben figyel a CLUSTERBUILD-üzenetekre, mert ha kap egy ilyen, akkor reagál rá egy IAMREADYTOJOIN-üzenettel. Ha erre válaszul egy INVITE-üzenetet is kap, nem próbál saját klasztert építeni, hanem csatlakozik a már meglévőhöz. Ebben az állapotban a modul mind két státusz LED-je ég.
- 3.) **Cluster build** : Ebben az állapotba akkor kerül a node, ha lejárt a belső számlálója és nem kapott meghívást. Ez esetben maga próbál klasztert építeni. Ha ez sikerül neki, a Normal process állapotba kerül. Ha nem, akkor az Outside cluster állapotba.
- 4.) **Normal process** : Amennyiben egy csomópont épített egy klasztert, vagy csatlakozott egyhez, megkezdheti a normál működést. Az állapot elérése után csak a zöld LED marad égve, az alkalmazásréteg pedig visszakapja a vezérlést.
- 5.) **Outside cluster** : Ha egy csomópontnak nem sikerült klasztert építenie és nem is hívták meg sehová, ebbe az állapotba kerül. Ilyenkor a piros LED marad égve, az alkalmazásréteg pedig nem kapja vissza a vezérlést. Az állapotból egy INVITE-üzenet képes átvinni a Normal process állapotba.



31. ábra. A modulok állapotátmeneti diagramja

A klaszterezés során háromféle üzenetet adnak a node-ok: CLUSTERBUILD-et, mellyel egy új klaszter alakulására hívják fel a figyelmet és mindenkinek szól, aki hallja, IAMREADYTOJOIN-t, mely a CLUSTERBUILD-re válasz és INVITE-üzenetet, amit csak azok kapnak meg, akiket meghívnak a klaszterbe.

8.2.4. A klaszterezés tesztelése

A klaszterezés tesztelése 10 csomóponttal történt, melyeknek háromtagú klasztereket kellett létrehozni. A tesztelés egyrészt a státusz LED-ek alapján, vizuálisan történt, másrészt a forgalom lehallgatásával. Ebben a fejezetben egy ilyen lehallgatás látható.

A lehallgató ugyan olyan modul típusú modul, mint a többi, csak egy kicsit módosított program fut rajta és a számítógéphez van kötve, ahová soros porton küldi ki a vett adatokat. Indulása után vár egy gombnyomást, aminek a hatására küld egy GO-tartalmú broadcast-üzenetet. Ez az összes node-ot feléleszti és így egyszerre kezdenek működni.

A kommunikációt megfigyeli és kiküldi soros portra. Egy ilyen lehallgatás eredménye látható a 7. táblázatban.

7. táblázat. *Kommunikáció klaszterezés közben*

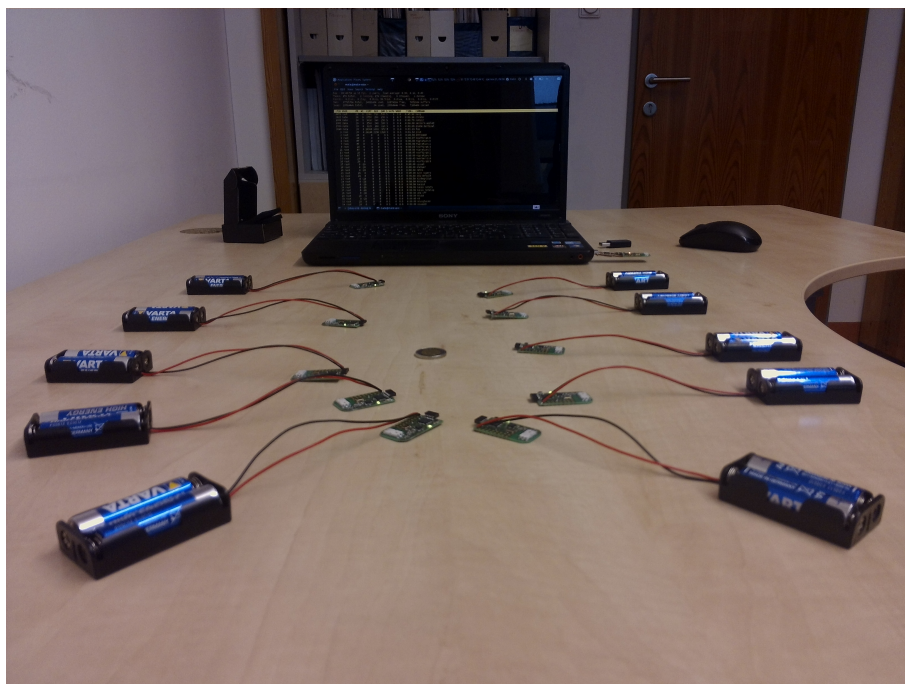
#	Feladó	Címzett	Üzenet
1	1111	BROADCAST	CLUSTERBUILD
2	7777	1111	IAMREADYTOJOIN
3	9999	1111	IAMREADYTOJOIN
4	5555	1111	IAMREADYTOJOIN
5	3333	1111	IAMREADYTOJOIN
6	2222	1111	IAMREADYTOJOIN
7	4444	1111	IAMREADYTOJOIN
8	1111	7777	INVITE11
9	1111	5555	INVITE11
10	2222	BROADCAST	CLUSTERBUILD
11	3333	2222	IAMREADYTOJOIN
12	9999	2222	IAMREADYTOJOIN
13	6666	2222	IAMREADYTOJOIN
14	4444	2222	IAMREADYTOJOIN
15	2222	3333	INVITE22
16	2222	9999	INVITE22
17	4444	BROADCAST	CLUSTERBUILD
18	6666	4444	IAMREADYTOJOIN
19	8888	4444	IAMREADYTOJOIN
20	4444	6666	INVITE44
21	4444	8888	INVITE44
22	AAAA	BROADCAST	CLUSTERBUILD

A log alapján látható, hogy a klaszterezés során a node-ok háromféle üzenetet küldenek egymásnak. Az első sorban látható, hogy az 1111 címmel rendelkező (ezt nem számként, hanem karaktersorozatként kell értelmezni) node-nak jár le először a számlálója, ezért küld egy broadcast-üzenetet, melyben jelzi a hallótávolságra lévőknek, hogy épít egy klasztert. (A csomópontok felprogramozása során szándékosan a fizikai cím alapján lettek meghatározva a számlálók.)

A másodiktól a hetedik sorig láthatóak azok a node-ok, akik erre válaszoltak egy IAMREADYTOJOIN üzenettel. A klaszterépítő csomópont eközben vár. Miután lejárt egy (fix méretű) belső számlálója, meghív két csomópontot az INVITEXY üzenettel. Ez látható a nyolcadik és kilencedik sorban. Az üzenet utolsó két bájtja a klaszter broadcast-címét határozza meg. Az üzenetet fogadó csomópontok (7777 és 5555) ezután megváltoztatják a címük első két bájtját (1177-re és 1155-re) és belépnek normál működési állapotba. Az így kialakult klaszter tagjai elérik egymást az 1100 címen.

A maradék hét, még mindig *Wait for build cluster* állapotban lévő node folytatja a klaszterezést. A következő BROADCAST-üzenetet a 2222-es node küldi. Ez a tizedik sorban látható. Erre az üzenetre is válaszolnak a csomópontok, amik közül a klaszterépítő a 3333-ast és a 9999-est hívja meg. Ezzel kialakult a második klaszter.

A harmadik klaszter kialakulása a tizenhetedik és a huszonegyedik sor között látható. Ekkorra a 10 node-ból 9 már tagja valamelyik klaszternek. A huszonkettedik sorban (utolsó üzenetként) az AAAA című node is küld egy broadcast-üzenetet, de már nincs ki feleljen rá, így ő nem képes klasztert építeni, ezért *Outside cluster* állapotba kerül. A csomópontok fizikai elrendezése a 32. ábrán látható.



32. ábra. A modulok fizikai elrendezése

A 7. táblázatban látható adatok ezzel az elrendezéssel készültek. A fénykép azután készült, hogy a klaszterek összeálltak. (A legtöbb node-on látszik is a zöld státusz LED.) Mivel a node-ok közel voltak egymáshoz, a CLUSTERBUILD-üzeneteket mindegyiknek hallania kellett. (A logban nem látszik az összes válasz a loggoló modul hibája miatt.)

8.2.5. További tervek

A klaszterezés jól működik, ami a hálózat alapja, de önmagában nem hordoz semmilyen funkcionalitást. Az ütemterv szerint a projekt mspgcc-re történő portolása után először a szemaforral kapcsolatos függvényeket (*takeSemaphore*, *releaseSemaphore*) kell megvalósítani, majd az adatíró és olvasó, végül a távoli függvényhívásokért felelős kódokat kell implementálni. Miután ezek megvannak, implementálni lehet a tényleges alkalmazásrétet is, ahol a hangminták feldolgozása történik.

8.3. Mobiltelefon, mint jeladó

A mobiltelefonra készítendő program feladata, hogy frekvenciamodulált hangjelet adjon ki, minél hosszabb ideig. Ez látszólag egyszerű és egyértelmű feladat, de rengeteg kérdés felmerülhet. Ebben a fejezetben egy Android-alapú okostelefonon alapuló megoldás problémái kerülnek ismertetésre, de a telefonra készítendő alkalmazás terjedelmi okokból nem kerül bemutatásra.

Egy földrengés után (ahogy arról már volt szó) megszűnnek a különböző hálózatok, a telefont pedig egy nagy erőhatás éri, amit a gyorsulásmérőkkel képes érzékelni. Android-platformra készíthető egy olyan, a háttérben futó alkalmazás, ami figyeli a térerősséget és a gyorsulásmérők adatait. Ha egy hirtelen gyorsulás után a térerősség megszűnik, akkor aktiválja magát és bekapcsolja a sípoló hangot.

Egy átlagos okostelefon akkumulátora átlagos használat mellett egy-másfél napot bír ki. Mivel a telefon jó eséllyel nincs feltöltve a katasztrófa bekövetkeztekor, az akkumulátorának élettartamát minél jobban ki kell nyújtani. Ehhez minden felesleges funkciót le kell

kapcsolni, mint pl. a GPS-t, GSM-modult, WiFi-t és a kijelzőt. A GSM-modult és a WiFi-t érdemes legalább félóránként visszakapcsolni arra az esetre, ha visszaállították a GSM-hálózatot. Ez esetben azon keresztül is lehet segítséget kérni.

A telefon feladata a hangmintázat generálása (400 és 500Hz között). Ezzel az egyébként is csapdába esett túlélőt még rosszabb lelkiállapotba juttathatja, hiszen a tiszta szinuszos hangok még nyugodt körülmények között is nagyon zavaróak. Éppen ezért nem szabad folyamatosan szólania a hangnak. A detektálást nem nehezíti, ha percenként csak 10 másodpercre kapcsol be, de a túlélőt (és az akkumulátort) kíméli vele.

A romok alatt rekedt felhasználó egy érintéssel jelezheti a telefonnak, hogy él. Ekkor a telefon egy másik hangmintázatot bocsájthat ki, ezzel jelezve, hogy tényleg van a közelében egy túlélő. Így a mentőegységek előnyben tudják részesíteni a biztos találatokat egy esetleges gazdátlan mobillal szemben.

9. Összefoglalás és a továbbfejlesztés lépései

A dolgozat során ismertetésre került egy olyan WSN-protokoll, amely a párhuzamos számításokat erősen támogatja. Egy ilyen protokollal működő WSN segítségével megvalósíthatóak olyan összetett és memóriaigényes mérési feladatok, amikre a klasszikus WSN-architektúra alkalmatlan. Ilyen feladat például a dolgozatban ismertetett hang alapú túlélődetektáló rendszer is, ahol több ezer csomópont mérési eredményét kell feldolgozni és elemezni másodpercenként. A rendszer mobiltelefonok által kibocsátott, frekvenciamodulált jelek detektálására képes (és így valójában a mobiltelefont keresi meg), az okostelefonok elterjedt használata miatt ez a hatékonyságát nem rontja [24].

A rendszert tovább is lehet fejleszteni úgy, hogy mobiltelefonok helyett ténylegesen a romok alatt rekedt túlélők hangját próbálja detektálni. Ez esetben biztosan nem lesz elég 10 csomópont kooperációja, hiszen az ehhez szükséges algoritmus sokkal komplexebb, de végső soron ez is visszavezethető a spektrogram elemzésére. Az ilyen irányú továbbfejlesztés előtt érdemes konzultálni egy akusztikai szakértővel is.

Magát a protokollt nem csak a katasztrófavédelem területén lehetne hatékonyan felhasználni. Több olyan terület is van, ahol kis fizikai méret és alacsony ár mellett van szükség nagy számítási kapacitásra. Ilyen területek például a robotika bizonyos részei, ahol cél lehet olyan kisméretű és olcsó robotok előállítása, amik egymással kooperálva oldanak meg különböző problémákat. Ilyenkor előfordulhatnak olyan mérési vagy döntési helyzetek, amikor a probléma bonyolultsága miatt szükség van a számítási kapacitás megosztására. Tipikusan ilyen terület a nanorobotika, ami jelenleg egy erősen kutatott terület.

Szintén ilyen terület az extrém kisméretű szenzorokból álló, ún. smart dust is [37]. A smart dust-projektek célja, hogy sikerüljön néhány köbmilliméteres szenzorokat gyártani, amik képesek mérni és kommunikálni egymással. Ilyen kis méretek mellett már a legegyszerűbb mérés elvégzésekor is kevés lehet az egyes csomópontok számítási kapacitása.

A protokollt nem csak a mérésekhez lehet felhasználni. A WSN-rendszerekben az optimális útvonal megtalálása általános esetben NP-teljes [38] probléma. Egy megfelelő Hopfield-hálózattal jó becslést lehet adni a megoldásra, ami egy könnyen párhuzamosítható algoritmus.

Ezekhez a feladatokhoz a protokollt is fejleszteni kell. A lehetséges továbbfejlesztések között szerepel a dinamikusan változó klaszterek és mozgó csomópontok támogatása, nem homogén (különböző funkciókat megvalósító) csomópontok kezelése, valamint szofisztikáltabb szemaforkezeléssel többfajta megosztott erőforrás támogatása is.

Hivatkozások

- [1] UNISDR. (2012. 10. 12). Disaster Statistics [Online] Available: <http://www.unisdr.org/we/inform/disaster-statistics>
- [2] PreventionWeb. (2012. 10. 13). Wild Fire - Data and Statistics [Online] Available: <http://www.preventionweb.net/english/hazards/statistics/?hid=73>
- [3] KSH. (2012. 05. 17). Személy sérüléses közúti közlekedési balesetek [Online] Available: http://www.ksh.hu/docs/hun/xstadat/xstadat_eves/i_ods001.html
- [4] Federal Emergency Management Agency. (1997). FIRE DEATH RATE TRENDS. [PDF] Available: www.usfa.fema.gov/downloads/pdf/statistics/v12i8.pdf
- [5] Harvard Sensor Networks Lab. (2012. 9. 14). CodeBlue: Wireless sensors for medical care. [Online] Available: <http://fiji.eecs.harvard.edu/CodeBlue>
- [6] AID-N. (2012. 9. 14). AID-N. [Online] Available: <http://www.jhuapl.edu/aid-n/>
- [7] Jason Palmer. (2011. 09. 12) Breath test could identify trapped disaster victims. [Online] Available: <http://www.bbc.co.uk/news/science-environment-14859492>
- [8] TAU GIKEN. (2012. 10. 03) Life detector. [Online] Available: www.taugiken.jp/tau/life_detector.pdf
- [9] BBC (2012. 10. 05) Earthquake rescue: How survivors are found. [Online] Available: <http://news.bbc.co.uk/2/hi/americas/8459653.stm>
- [10] R. Jurdak, "Ad Hoc and Sensor Networks: Opportunities and Challenges", in Wireless Ad Hoc and Sensor Networks, Dublin, Ireland: Springer, 2007
- [11] Y. Yu, et al., "Evolution of Sensor Nodes" in Information processing and routing in wireless sensor networks, World Scientific Publishing Co. Pte. Ltd. 2006
- [12] Edward C. Whitman. (2012. márc. 3) SOSUS [online]. Available: http://www.navy.mil/navydata/cno/n87/usw/issue_25/sosus.htm
- [13] XBeeTM/XBee-PROTM OEM RF Modules, 1 ed. MaxStream, Lindon, 2006, pp.5-6
- [14] Libelium. (2010. aug. 20) Forest Fire Detection using Wireless Sensor Networks [video] Available: <http://www.youtube.com/watch?v=pH0JPWk0mGA&feature=related>
- [15] Libelium. (2012. május 8). About us [online]. Available: <http://www.libelium.com/company/about>
- [16] MTI. (2007. aug. 10). Intelligens kamerával az erdőtűz ellen [online]. Available: http://hirposta.hu/cikk/246541/90_embert_valt_ki_az_erdotuz_figyelo_kamera/
- [17] Libelium. (2011. okt. 26) Vehicle Traffic Monitoring Platform with Bluetooth Sensors over ZigBee. [online]. Available: <http://www.libelium.com/development/articles/112992991737>
- [18] M. Lőrincz, "Sentinel box monitoring rendszer" Technika m, vol. 55, no. 2, pp. 26-27, Mar. 2012
- [19] Y. Yu, et al., 1.5 in Information processing and routing in wireless sensor networks, World Scientific Publishing Co. Pte. Ltd. 2006

- [20] R. Jurdak, "7.3 Why Cross-Layer Design for Ad Hoc and Sensor Networks?", in Wireless Ad Hoc and Sensor Networks, Dublin, Ireland: Springer, 2007
- [21] D. Gislason, "ZigBee Is Highly Reliable" in Zigbee wireless networking
- [22] H. Monson. (1999. dec. 14.) Bluetooth Technology and Implications [Online]. Available: <http://www.sysopt.com/features/network/article.php/3532506>
- [23] H. Monson. (1999. dec. 14.) Bluetooth Technology and Implications [Online]. Available: http://www.sysopt.com/features/network/article.php/12029_3532506_2
- [24] MTI (2012. 11. 17) Magyar fejlesztésben az életmentő eszköz. [Online] Available: <http://www.hir24.hu/kulfold/2012/11/17/magyar-fejlesztésben-keszul-az-eletmento-eszkoz/>
- [25] Guoqiang Maoa (2007. 07. 11) Wirelesssensornetwork localization techniques [PDF] Available: <http://www.sciencedirect.com/science/article/pii/S1389128606003227>
- [26] Kolozs Sándor (2006. 05. 09) Hang elmélet. [Online] Available: http://www.szsz.hu/wiki/Hang_Elmélet
- [27] A. V. Oppenheim, R. W. Schaffer, "4.2 Frequency-domain representation of sampling" in Discrete-Time Signal Processing, 3th ed. London, England, Pearson
- [28] A. V. Oppenheim, R. W. Schaffer, "8.5 Fourier representation of finite-duration sequences: the DFT" in Discrete-Time Signal Processing, 3th ed. London, England, Pearson
- [29] A. V. Oppenheim, R. W. Schaffer, "10.3 The time-dependent Fourier transform" in Discrete-Time Signal Processing, 3th ed. London, England, Pearson
- [30] A. V. Oppenheim, R. W. Schaffer, "9. Computation of the Discrete Fourier Transformation" in Discrete-Time Signal Processing, 3th ed. London, England, Pearson
- [31] Nancy Ann Lynch, "Előszó az angol nyelvű kiadáshoz" in Osztott algoritmusok, Budapest, Magyarország, Kiskapu Kiadó
- [32] Nancy Ann Lynch, "Általános, szinkron hálózatok algoritmusai" in Osztott algoritmusok, Budapest, Magyarország, Kiskapu Kiadó
- [33] Nancy Ann Lynch, "Alsó korlát nem összehasonlítás-alapú algoritmusok üzeneteinek számára" in Osztott algoritmusok, Budapest, Magyarország, Kiskapu Kiadó
- [34] Xiao, Wei, Yaqing Tu, and Hua Xu. "A hierarchical routing protocol to handle topological variation for wireless sensor network." Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on. IEEE, 2008.
- [35] NICTA (2012. 3. 17) What is Castalia? [Online] Available: <http://castalia.npc.nicta.com.au/>
- [36] TI (2012. 08. 13) Product Information [Online] Available: <http://www.ti.com/tool/msp430-3p-awgh-amb8423-devbd>
- [37] Kris Pister (2010.). Smart Dust [Online] Available: <http://robotics.eecs.berkeley.edu/~pister/SmartDust/>
- [38] Jia, Xiaohua and Wu, Jie and He, Yanxiang. (2005) "Minimum Data Aggregation Time Problem in Wireless Sensor Networks", Springer Berlin Heidelberg