

Vezeték nélküli kommunikációs protokollok tervezése és
implementációja elosztott ipari vezérlőrendszerekhez
c. TDK dolgozat

Készítette: Lőrincz Máté
Mérnök informatikus, 2. évfolyam, MSc

Témavezető: Dr. Oláh András, egyetemi docens



Pázmány Péter Katolikus Egyetem
Információs Technológiai és Bionikai Kar
2014

Tartalomjegyzék

1. Bevezetés és motiváció	3
2. Tradicionális ipari vezérlőrendszerek és vezeték nélküli kommunikációs technológiák	5
2.1. Tradicionális ipari vezérlés	5
2.2. Vezeték nélküli hálózatok ipari környezetben	11
2.3. Ipari környezetbe szánt kísérleti kommunikációs protokollok	14
3. Új, elosztott ipari vezérlést megvalósító vezeték nélküli keretrendszer	18
3.1. A végpontok hardver és szoftver komponensei	18
3.2. A kommunikációs protokoll	25
3.3. Fejlesztőkörnyezet	33
4. Esettanulmány - Automatizált felrakógép	37
4.1. A helyszín bemutatása	37
4.2. A működés leírása	38
4.3. A vezérlőréteg	39
4.4. Vezérlés megvalósítása TinyPLC rendszerrel	39
5. Szimuláció és validáció	42
5.1. A szimuláció célja	42
5.2. Szimulátorok bemutatása	43
5.3. Saját szimulátor bemutatása	43
5.4. Protokoll tesztelése a szimulátorral	45
6. Tovább lépési lehetőségek	49
A. Letölthető tartalom	50

Absztrakt

Az elmúlt évszázadokban a különböző iparágakban egyre inkább megjelent az igény az emberi munkaerő kiváltására. Bár kezdetben a fő motivációs erő a költségek csökkentése és a hatékonyság növelése volt, mára már kialakultak olyan területek, melyek a megfelelő gépesített gyártósorok nélkül a mai minőségi követelményeket nem tudnák teljesíteni. Tipikus példa erre az atomipar, ahol a veszélyes anyagok és a szigorú biztonsági előírások miatt nélkülözhetetlen az automatizált vezérlés, de ma már ide sorolható minden olyan iparág, ahol szempont a megbízható, egységes minőség, valamint a teljes nyomonkövethetőség. (Tipikus példa az autóipar és gyógyszeripar.)

A gyártósorok teljes, valós idejű vezérlését kezdetben huzalozott vezérlőpanelek segítségével oldották meg, de mára ezeket leváltották a programozható vezérlők, vagyis a PLC-k, melyek olyan, sok ki- és bemenettel rendelkező, valós idejű számítógépek, melyeket mind hardver, mind szoftver szempontjából az ipari körülmények és igények szerint fejlesztettek ki.

Egy átlagos ipari létesítményben nem csak egy PLC gondoskodik a folyamatok megfelelő irányításáról és felügyeletéről, hanem több vezérlő és a hozzájuk tartozó ki- és bemeneti kártyák együttesen, egy közös (általában ethernet-alapú) hálózatba kötve határozzák meg a termelés menetét. Bár minden ilyen létesítmény különböző, az architektúrájuk nagyon hasonló: a vezérlés egy központi helyiségben található és egy nagy sebességű hálózaton keresztül irányítja a tényleges beavatkozást végző, valamint a visszajelzéseket fogadó ki- és bemeneti kártyákat.

A TDK dolgozat középpontjában egy olyan, vezeték nélküli, elosztott vezérlést megvalósító alkalmazás áll, mely képes ellátni egy gyártási folyamat monitorozását és vezérlését. A rendszer alapját olyan kiskapacitású, olcsó vezérlőegységek képezik, melyek teljesítménye alulmarad ugyan egy átlagos PLC-hez képest, de egy részfolyamat vezérlését képesek elvégezni. Ezek az eszközök a klasszikus architektúra ki- és bemeneti kártyáival szemben szigetüzemű működésre képesek, így a feladatukat minden mástól függetlenül, nagyobb megbízhatósággal képesek elvégezni. (A kommunikációs hálózat hibái nem okoznak semmilyen fennakadást a vezérlésben.)

Természetesen a gyártás részfeladatai között bizonyos információkat át kell adni, amire a dolgozat során egy erre a célra kifejlesztett kommunikációs protokoll kerül bemutatásra. A tervezés során a fő szempont a megbízhatóság és kiszámíthatóság volt. A kifejlesztett vezérlőrendszer használatához szükség van egy megfelelő szoftvercsomagra, mely segítségével a fejlesztők képesek dolgozni a bemutatott rendszerrel és megbízható, könnyen nyomon követhető programokat tudnak előállítani.

A dolgozat részletesen bemutatja a vezérlést megvalósító végpontok szoftveres és hardveres felépítését, az ezeket összekötő, vezeték nélküli protokollt, az eredményeket validáló szimulációt valamint a fejlesztőknek szánt szoftvercsomagot az elkészült prototípusokon keresztül.

1. Bevezetés és motiváció

Az elmúlt 50 évben az automatizálási iparág – a PLC-knek köszönhetően – teljesen átalakult. A régebbi, relé-alapú megoldásokat mára már teljesen leváltották a programozható eszközök, melyek kapacitása és funkcionalitása széles skálán mozog, de a felépítésük és működési elvük hasonló.

A legtöbb ipari létesítményben egy PLC egyszerre nagyon sok, egymástól csak kismértékben függő eszközt vezérel. Ez azt eredményezi, hogy ha az egyik eszköz működésében valami probléma lép fel, akkor a hiba továbbterjedhet, és az kihat a gyártás más területeire is, felesleges leállásokat – és óriási anyagi károkat – okozva ezzel. A felesleges leállások mellett a vezérlőhálózat meghibásodása is problémát okozhat. Ha a PLC és a végpontjai között megszűnik a kommunikáció, nem lehet garantálni a leszakadt eszköz szabályos leállítását, ami akár azonnali meghibásodást is okozhat. (Ezt a problémát jelenleg hardveres megoldásokkal szokták kivédeni.)

Jelen dolgozat célja egy olyan rendszer bemutatása, mely a jelenleg elterjedt módszerrel szemben a vezérlőlogikát szétesztja és a vezérlendő eszközök közelébe viszi, megszüntetve ezzel a meghibásodásból és a kapcsolat szakadásából adódó hibalehetőségeket. Egy ilyen elosztott rendszer a működését tekintve sokkal rugalmasabb lenne a jelenlegi megoldásoknál, ami ugyanakkor két problémát is felvet:

- a **fejlesztők dolga sokkal nehezebb**, hiszen egyszerre kell karbantartaniuk egy több száz eszközből álló rendszert;
- a **kommunikációs hálózat topológiája is változik**, mert miközben a központi vezérlés esetén minden végpont egy PLC-vel kommunikált, addig az új rendszerben már egymás között kell intenzív kommunikációt folytatniuk.

A fenti problémák miatt nem elég csak a beavatkozást végző végpontokat újratervezni, szükség van egy megfelelő **kommunikációs protokollra** és egy **fejlesztőkörnyezetre** is.

A dolgozat felépítése során az volt a célom, hogy bemutassam a folyamatot, ami a rendszer megtervezéséhez és implementálásához vezetett. A dolgozat fejezet szintű felépítése a következő:

- az **1. fejezet (Bevezetés és motiváció)** – ismerteti a technológiai motivációkat és a dolgozat célkitűzéseit;
- a **2. fejezet (Tradicionális ipari vezérlőrendszerek és vezetékek nélküli kommunikációs technológiák)** – néhány kísérleti technológia mellett – az eddig elterjedt módszerek kerülnek bemutatásra;
- a **3. fejezet (Új, elosztott ipari vezérlést megvalósító vezetékek nélküli keretrendszer)** bemutatja a dolgozat során elkészült rendszert a végpontokkal, kommunikációs protokollal és fejlesztőkörnyezettel együtt;
- a **4. fejezet (Esettanulmány - Automatizált felrakógép)** egy esettanulmányt tartalmaz, mely javaslatokat tesz a dolgozatban bemutatott rendszer alkalmazására;
- az **5. fejezet (Szimuláció és validáció)** különböző szimulációkat mutat be, valós mérésekkel alátámasztva;

- végül a **6. fejezet (Tovább lépési lehetőségek)** összefoglalja az elért eredményeket és kijelöli a tovább lépési irányokat.

2. Tradicionális ipari vezérlőrendszerek és vezeték nélküli kommunikációs technológiák

A különböző iparágakban nagyon fontos a megbízható minőségű, kiszámítható és nyomonkövethető gyártás. Ennek érdekében az elmúlt évtizedekben komoly erőfeszítések történtek, hogy az egyes ipari rendszerek tervezését és fejlesztését egyszerűsítsék és szten-dardizálják. Ennek eredményeként az ipari vezérlést speciális vezérlőkre, úgynevezett PLC-kre (Programmable Logic Controller) bízák, melyeknek megvannak a saját programozási szabványaik, azok előnyeivel és hátrányaival.

A PLC-k mellett természetesen megjelentek más jellegű ipari megoldások is, mint például a National Instruments Real-Time Industrial Controllers [1] vezérlőcsaládja, vagy a Texas Instruments Industrial System Solutions [2] megoldásai. Ezeknek az eszközöknek a használati területe, felépítése és a programozása eltér a logikai vezérlőktől, a dolgozatban csak érintőlegesen lesz róluk szó.

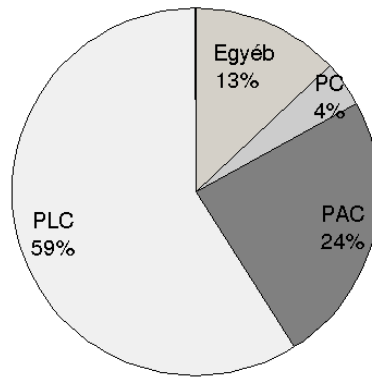
2.1. Tradicionális ipari vezérlés

Az 1960-as évek végéig az ipari folyamatokat huzalozott kapcsolószekrények vezérelték, minden logika hardveresen, reléekkel volt megvalósítva [3]. Ez azt eredményezte, hogy a gyártási technológia legkisebb módosítása az elektromos szekrények újratervezését és újrahunalozását vonta maga után. (Természetesen ehhez a termelést is le kellett állítani.) Ez nagyban növelte az új termékek piacra lépésének idejét és kockázatát.

A felmerülő problémák megoldásának érdekében 1968-ban a General Motors mérnökei programozható eszközökkel szerették volna lecserélni az addig használt relés megoldásokat. Ennek érdekében négy tervezési kritériumot fogalmaztak meg [3]:

- i.) legyen **könnyen programozható**, mint egy számítógép, de árban maradjon versenyképes az addig használt relé-alapú rendszerekkel;
- ii.) az akkor már (dokumentációs célokra) általánosan használt **létradiagram** segítségével könnyen lehessen programozni;
- iii.) legyen **ellenálló** az ipari körülmények között előforduló szennyeződésekkel, nedvességgel, elektromágneses zajokkal és vibrációval szemben;
- iv.) legyen **moduláris** és bővíthető.

A megfogalmazott kritériumok alapján több cég is megkezdte a saját termékének fejlesztését, melynek következtében 1969-ben már kereskedelmi forgalomba kerülhettek az első PLC-k. Napjainkra már (ahogy az az 1. ábrán is látszik) az ipari rendszerek túlnyomó része PLC-alapú [4].



1. ábra: Az ipari vezérlők piaci megoszlása. A grafikonon látható, hogy az ipari szereplők többsége tisztán PLC-alapú rendszereket használ (59%), negyede (24%) pedig programozható automatizálási vezérlőket (PAC [5]), melyek – válaszul az egyre bonyolultabb vezérlési feladatokra – a PC-k és PLC-k integrálásából születtek.

2.1.1. A PLC-k felépítése

A legtöbb PLC felépítése – összhangban az 1968-ban megfogalmazott elvekkel – a mai napig moduláris (2. ábra). Az eszköz alapja egy váz, melyben egy processzor, néhány kommunikációs kártya, valamint ki- és bemeneti kártyák találhatók.



2. ábra: Példa a PLC-k moduláris felépítésére. A bal oldali ábrán egy Rockwell, jobb oldalon pedig egy Siemens PLC látható. Mindkettő alapja egy váz, melybe tetszőleges funkciójú kártyákat lehet tenni (pl.: 3. és 4. ábra).

A **processzor** (3. ábra) a PLC legfontosabb része. Felelős a program tárolásáért és végrehajtásáért valamint a többi kártya vezérléséért. Egy vázban több processzor is helyet kaphat, bár ez nem gyakori megoldás. A modernebb processzorokat Etherneten keresztül lehet programozni, de előfordul, hogy a programozáshoz soros port vagy valamilyen speciális kábel szükséges.

A processzor mellett szinte mindig megtalálható egy vagy több **kommunikációs kártya**. Rockwell esetén ez leggyakrabban Ethernet, ControlNet vagy DeviceNet-alapú. Ezekon a kártyákon keresztül tartja a processzor a kapcsolatot a fejlesztőeszközzel, az operátorpulttal, a terepen lévő többi PLC-vel és a kihelyezett IO modulokkal.

A gyártás szempontjából a PLC-k legfontosabb részei a ki- és bemenetei (4. ábra). Amennyiben a PLC közel található a vezérelendő géphez, elég a processzor mellé berakni a megfelelő típusú IO kártyákat, bár ez a ritkább eset. Sokkal jellemzőbb az olyan felépítés, melyben a processzor és a kommunikációs kártyák egy klimatizált szerverteremben vannak, az eszközöket pedig távoli IO modulokon keresztül vezérlik. Ennek a módszer-



3. ábra: Egy vázba pattintható kártya. A képen egy Rockwell 1785-L40B Series C PLC-5/40 Processzor modul látható. Az előlapján látható kulcsos kapcsoló az üzemmód kiválasztását szolgálja. A legtöbb PLC három üzemmóddal rendelkezik: programozási üzemmódban a felprogramozott kód inaktív, de módosítható, futási üzemmódban a kód aktív, de írásvédett, távoli üzemmódban pedig a fejlesztőkörnyezeten lehet választani az előbbi két mód közül.

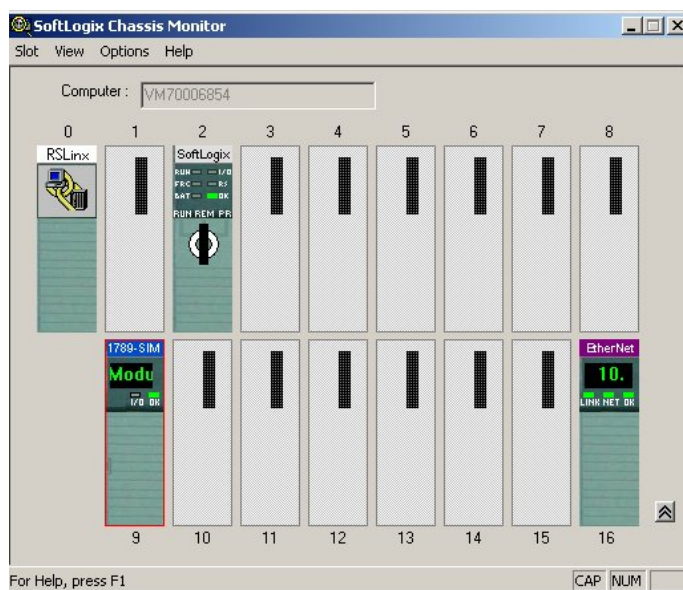
nek az előnye, hogy egy processzor egymástól távol elhelyezkedő eszközöket is vezérelhet, hátránya pedig, hogy a vezérlés során számolni kell egyrészt a kommunikáció késleltetésével, másrészt pedig fel kell készülni egy esetleges vezetékszakadásra. (A legtöbb távoli IO modul támogatja a redundáns vezetékeezést.) Az IO modulok nem képesek semmilyen vezérlésre, csak a processzor utasításait hajtják végre.



4. ábra: IO modul és IO kártyák összehasonlítása. A bal oldali ábrán egy Etherneten keresztül vezérelhető fejegység és a hozzá csatlakoztatott kártyák (6 db) láthatóak. Felépítése hasonlít a PLC-hez, de nem tartalmaz processzort. Jobb oldalon egy váz látható, benne tíz IO kártyával és egy processzorral. Jobbról a második és a harmadik hely üres a vázban, ez nem okoz semmilyen problémát.

Vannak olyan PLC-k is, amikben a processzor egybe van építve a táppal, valamilyen kommunikációs kártyával és különböző ki- és bemenetekkel. Ezek a specializáltabb eszközök jellemzően kisebbek és olcsóbbak, cserébe a felhasználónak le kell mondania a testreszabhatóságról. Példa erre a Rockwell MicroLogix termékcsaládja [6].

Érdeemes még megemlíteni a **virtuális PLC**-ket, melyekre jó példa a Rockwell SoftLogix (5. ábra). A SoftLogix esetén a felhasználó kap egy vázat (melynek mérete állítható 10 és 17 között) és belerakhat (megfelelő licenz birtokában) tetszőleges számú processzort, Ethernet kártyát és IO kártyákat. Az IO kártyáknak nincs semmilyen valós szerepük, a virtuális Ethernet-kártyán keresztül viszont van lehetőség valódi IO modulokra csatlakozni, valamint a processzor képes használni a számítógép fizikai soros portjait is. Mivel a PC-s megvalósítás miatt a virtuális PLC nem lesz sem valós idejű, sem ellenálló felépítésű, az ilyen emulált PLC-eket általában csak fejlesztés és tesztelés közben használják.



5. ábra: Rockwell SoftLogix. Az ábrán látható szoftverben egy virtuális processzor (2. slot), egy szintén virtuális IO kártya (9. slot) és egy Ethernet-kártya (16. slot) van használva. A feltöltött program a virtuális processzorban fut, az Ethernet-modulon keresztül képes kommunikálni a külvilággal.

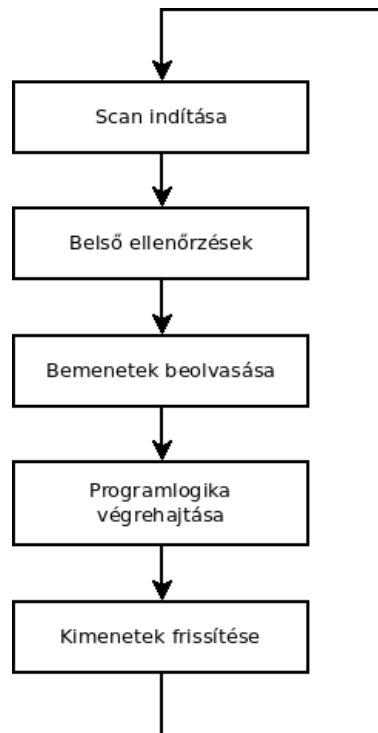
2.1.2. A PLC-k programozása

A különböző PLC-k programozása erősen gyártófüggő. Minden gyártó biztosít valamilyen fejlesztőeszközt és programozási módot. Utóbbi történhet például etherneten keresztül vagy valamilyen speciális kábellel. Két különböző gyártó fejlesztőkörnyezete nem kompatibilis egymással, de az IEC 61131-3 [7] szabványnak köszönhetően az elérhető programozási nyelvek típusától és gyártótól függetlenül hasonlóak egymáshoz.

A PLC-k programozása nehézséget okozhat egy asztali programozásban jártas fejlesztő számára, mert más szemléletmódot igényel. Míg egy PC-s programban az utolsó utasítás végrehajtása után leáll a program, addig egy PLC esetén a fejlesztő által megírt kódsorok folyamatosan futnak egy végtelen ciklusban [8]. (Ez a ciklus a beolvasási ciklus, angolul scan cycle vagy operating cycle.) Mivel egy PLC jellemzően több eszközt is kezel és nincs benne szálkezelés, ezért a fejlesztő nem használhat blokkoló és várakozó utasításokat, mert azzal a beolvasási ciklus időtartamát túlságosan megnövelné. (A ciklus jellemzően ezredmásodpercekben mérhető, nagyon ritkán megy egy másodperc fölé. A túl hosszú beolvasási ciklus veszélye, hogy a PLC reakcióideje lelassul, a különböző szabályozók hatásfoka romlik.) Egy tipikus PLC utasításciklusa a 6. ábrán látható.

A legelterjedtebb nyelv a **létradiagram**[9] (angolul Ladder logic), melyet eredetileg azért alkottak meg, hogy a kapcsolószekrények kapcsolását könnyen és szabványosan meg lehessen tervezni. Programozási nyelvvé csak a '60-as évek végére alakult át. Bár a nyelv nagyon egyszerű, tetszőleges vezérlés megvalósítható benne. Előnyei, hogy grafikus, ezért könnyen tanulható és az értelmezéséhez nem szükséges mély programozási ismeret. A programozási módszer nevét a program létraszerű felépítéséről kapta. Minden programot egy vagy több létrafok (továbbiakban rung) épít fel. A rungok egymás után futnak le meghatározott sorrendben, de egymástól függetlenül. Egy rung utasításai balról jobbra hajtódnak végre.

A programozás során fontos fogalom a rung feltétel (rung condition). Ez – hasonlóan a kapcsolószekrényekben az áramhoz – a rung elején mindig igaz, de az utasítások hamissá



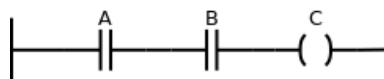
6. ábra: PLC utasításciklus. A PLC-k túlnyomó többsége az ábrán látható lépéseket hajtja végre másodpercenként akár több százszor. Van néhány kivétel is, például a Rockwell PLC-k a ki- és bemeneteket aszinkron frissítik, a programlogika végrehajtása közben.

fordíthatják (mint például egy szakadás). Ha egyszer hamis lett, többet nem lehet igaz. Ha a rung feltétel hamissá változott, a további utasításoknak általában nincs hatása. (Kivéve például az értékadó utasítást.) A legfontosabb utasítások az 1. táblázatban találhatóak.

1. táblázat: A létradiagram alapvető utasításai. Az utasítások képei a Rockwell RSLogix nevű fejlesztőkörnyezetéből származnak.

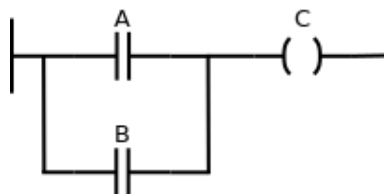
Utasítás neve	Utasítás képe	Jelentése
	Examine if Closed	Bemeneti paramétere egy bit. Ha az a bit igaz, és a rung feltétel igaz tőle balra, akkor jobbra is igaz lesz.
	Examine if Open	Bemeneti paramétere egy bit. Ha az a bit hamis, és a rung feltétel igaz tőle balra, akkor jobbra is igaz lesz.
	Output Energize	Értékadó utasítás. A bemenetként megadott bit a rung feltétellel lesz egyenlő.
	Output Latch	Értékadó utasítás. Ha a rung feltétel igaz, akkor a bemenetként megadott bit igaz lesz, egyébként változatlan marad.
	Output Unlatch	Értékadó utasítás. Ha a rung feltétel igaz, akkor a bemenetként megadott bit hamis lesz, egyébként változatlan marad.
	Elágazás	Ha az elágazás előtt a rung feltétel igaz és bármelyik elágaztatott létrafok végén is igaz marad, akkor az elágazás után is igaz lesz.
	Relációs vizsgálat	Példa numerikus utasításokra. Ha a bementi rung feltétel igaz és a relációs vizsgálat is igaz, akkor az utasítás után is igaz marad a feltétel.

A programozás során ezekből az alapelemekből lehet felépíteni a vezérlést. A 7. ábrán egy logikai ÉS kapcsolatot megvalósító rung látható. *A*, *B* és *C* logikai típusú paraméterek (bitek, Bool TAG-ek, a megnevezés gyártónként különbözik). Az első, *A*-t tartalmazó szimbólum bemenete (bal oldala) mindig igaz, mert előtte nincs semmi más. A XIC utasításnak akkor lesz a kimenete (jobb oldal) igaz, ha a bemenete igaz és a paraméterként megadott érték igaz. A második XIC kimenete akkor lesz igaz, ha az első XIC kimenete igaz és *B* igaz. A *C* alatti utasítás neve OTE (Output Energize). Ez egy értékadó utasítás, *C* értékét igazra állítja, ha a bemenete igaz, hamisra, ha a bemenete hamis.



7. ábra: Logikai ÉS kapcsolat. *C* akkor lesz igaz, ha *A* és *B* igaz.

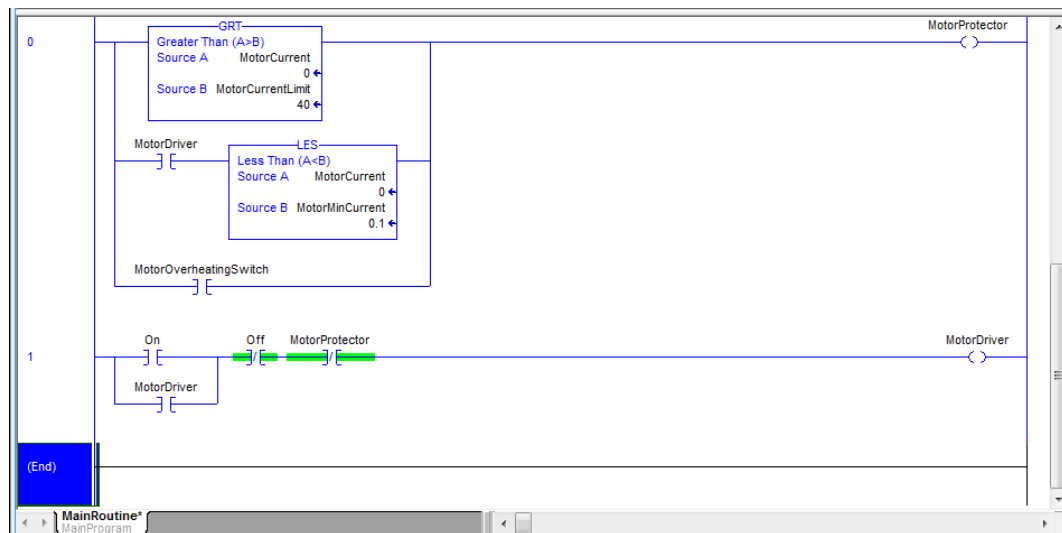
A 8. ábrán egy logikai VAGY kapcsolat látható, mely egy elágazással van megvalósítva. Egy ágban bármennyi elágazás lehet, az egyetlen követelmény, hogy mindegyiknek vissza kell térnie a főágba.



8. ábra: Logikai VAGY kapcsolat. *C* akkor lesz igaz, ha *A* vagy *B* igaz.

A 9. ábrán egy motorvezérlés látható. Az On és az Off fizikai gombok, melyek az operátorpanelen helyezkednek el. (A PLC-k a fejlesztő elől általában elrejtik a különbséget a fizikai ki- és bemenetek, valamint a csak a memóriában létező paraméterek között.)

Amikor az operátor megnyomja az On gombot, a PLC a következő beolvasási ciklusban MotorDrive értékét igazra állítja. (Mivel Off hamis és MotorProtector hamis.) Miután az operátor elengedi az On gombot, már MotorDriver igaz, tehát az az utáni beolvasási ciklusban is marad igaz, mert On-nal MotorDriver párhuzamos. (Hasonlóan az asztali nyelvekhez, egy paraméter értéke függhet saját korábbi értékétől. Ez a kódolási minta nagyon gyakori a létradiagramban, öntartásnak hívják [10].) Amennyiben az operátor megnyomja az Off gombot, az mindenképpen hamisra állítja MotorDriver-t. A motor akkor is leáll, ha a MotorProtector paraméter igazra vált az első rung alapján. Ez akkor történik meg, ha a motor áramfelvétele túl magas, túl alacsony (ez szenzorhibára utalhat), vagy a hőkapcsoló bekapcsol.



9. ábra: Motorvezérlő logika Az ábrán látható motorvezérlő két rungból áll: az első a motor lehetséges hibáit gyűjti össze egyetlen logikai paraméterbe, a második pedig a motor fizikai kimenetét vezérli.

A létradiagram egyik legnagyobb előnye, hogy futás közben könnyen látható a benne írt program működése. Például a XIC (Examine if Closed) utasítások zöldre váltanak, ha a paraméterük igaz, ahogy az értékadó utasítások is.

A **struktúrált szöveg** hasonlít a legjobban a klasszikus programozási nyelvek esetén megszokott módszerhez. Rockwell esetén egy egy Pascal-szerű nyelv, Siemens ehelyett az utasításlistát valósította meg. Előnye, hogy sokkal tömörebb, mint a létradiagram, érdemes ezt használni, ha komplex matematikai műveleteket vagy valamilyen bonyolult, többszintű döntést kell megvalósítani. Hátránya, hogy nehéz futás közben nyomon követni a működését.

Bár nem számít jó gyakorlatnak, előfordulhat, hogy a vezérlőlogikában ismétlőciklust vagy elől-, illetve hátultesztelő ciklusokat kell használni. Erre a létradiagram nem alkalmas, ilyenkor a fejlesztő kénytelen struktúrált szövegben dolgozni. A ciklikus kódrészletek a PLC programozása során kerülendőek, mert jelentősen lassíthatják az utasításciklust. Csak nagyon ritkán, speciális esetekben van rájuk szükség.

2.2. Vezeték nélküli hálózatok ipari környezetben

Egy gyártósor üzemeltetése során a robosztus vezérlés mellett nem szabad elfeledkezni a megbízható kommunikációs módszerekről sem. Egyszerűbb esetekben és rövid távolságokon megoldás lehet, ha a vezérlésben részt vevő modulok digitális ki- és bemeneteken

keresztül kommunikálnak egymással, de ez egy nehezen bővíthető, merev megoldást eredményez. Sokkal jobb gyakorlat választani egy kiforrott kommunikációs technológiát, amit a használt vezérlőeszközök támogatnak. A különböző gyártók kínálata nagyon széles.

A megfelelő protokoll kiválasztása a fizikai réteg meghatározásával kezdődik. Ehhez figyelembe kell venni az üzemcsarnok lehetőségeit (érdemes használni a meglévő vezetékeztést, amennyiben van), a használati igényeket (kommunikációban részt vevő eszközök száma, adatok nagysága), az áthidalandó távolságot és az alkalmazás igényeit (késleltetés, megbízhatóság). Általánosságban elmondható, hogy a különböző ipari területeken inkább a vezetékes megoldásokat preferálják, ha pedig vezeték nélküli összeköttetést kell létesíteni két eszköz között, akkor általában WLAN-t (31%), Bluetooth-t (22%) vagy ZigBee-t használnak [11]. Az elterjedtségük miatt először ezekről érdemes szót ejteni.

2.2.1. A ZigBee kommunikációs protokoll

A ZigBee kifejlesztése során egy olyan nyílt protokoll kidolgozása volt a cél, mely az alábbi követelményeknek felel meg [12]:

- nagy hálózatok üzemeltetése, 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, nyílt protokoll, melyet több gyártó is felhasználhat.

A ZigBee az OSI rétegek szerint az alábbiak szerint [12] épül fel:

– 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. A CSMA-CA hátránya, hogy a közeghozzáférési réteg nemdeterminisztikus [13], hiszen megjósolhatatlan, hogy hányadik küldési kísérlet fog sikerrel járni.

– 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 alternatív utat.

– **Alkalmazásréteg**

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

- i.) **Koordinátor (ZC):** A hálózat felépítéséért és vezérléséért felelős, általában ez a nyelő. A mérések ide futnak be. Kitüntetett címe (0x0) van a hálózatban.
- ii.) **Router (ZR):** A hálózat határainak kibővítéséért felelős, képes más eszközöket fogadni.
- iii.) **End device (ZE):** Általában ez végzi a méréseket, ugyanakkor nem képes további eszközöket kiszolgálni. Ez az egyetlen tagja a hálózatnak, ahol engedélyezett az alvási üzemmód, aminek azért van nagy a jelentősége, mert így akár évekig is képes üzemelni egy pár AA-elemmel.

A ZigBee egy jól átgondolt, a gyakorlatban már bizonyított, nyílt szabvány, mely általában jó választás egy adatgyűjtő WSN számára. 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, viszont egy vezeték nélküli ipari vezérlőrendszer szabványához nem ideális választás.

2.2.2. A Bluetooth kommunikációs protokoll

A Bluetooth eredetileg mobil és fix eszközök nagy sebességű és fokozott biztonságú összeköttetésére jött létre [14]. 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 ipari feladatok kiszolgálására fejlesztették ki: nem tesz lehetővé tetszőlegesen nagy hálózatokat (egy piconetben egy mester és 7 szolga szerepkörű eszköz lehet egyszerre jelen) és csak a csillagtopológiát támogatja. Ettől függetlenül hála a PAN-alapú rendszerek elterjedésének, napjaink egyik meghatározó vezeték nélküli technológiája.

A Bluetooth fizikai rétege 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.

Az adatkapcsolati rétege egy időosztásos csomagalapú protokoll mester-szolga struktúrával [15]. A mester (minden piconetben egy) egy szinkronizációs órajelet küld minden 312.5 μ s-ban. Két óraütés meghatároz egy 625 μ s-nyi időrést. Ekkora időrésekre kell felosztani egy csomagokat. Egy réspár hossza 1250 μ 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.

A Bluetooth protokollja lehetőséget biztosít úgynevezett scatternet létrehozására is, ami több piconet összekapcsolásával jön létre. Ilyenkor bizonyos eszközök az egyik piconet-

ben mestert, a másikban pedig szolga szerepkörben működnek. Csak kevés alkalmazás használja ki ezt a funkciót.

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.

A protokoll 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 legelterjedtebben használt szabvány.

2.2.3. A WiFi kommunikációs protokoll

A WiFi a 802.11 nyílt rádiós szabványra épülő kommunikációs protokoll [16]. A szabványnak több változata létezik (a, b, g, n), melyek sebességben és a használt frekvenciasávokban különböznek egymástól.

A 802.11-es protokollokról általánosan elmondható, hogy a rádiós közeghez történő megosztott hozzáférést – az Ethernet-hez hasonlóan [17] – CSMA (Carrier Sense Multiple Access) segítségével valósítják meg. A CSMA során a rádiós eszközök adás előtt belehallgatnak a csatornába. Amennyiben az szabad, megkezdhetik a saját csomaguk sugárzását.

A CSMA közeghozzáférési módszer segítségével lehetséges a csomagok prioritás szerinti kezelése (a késleltetésre érzékeny csomagokat az eszközök gyakrabban küldhetik), de nem lehet garantálni a maximális késleltetésre egy felső korlátot. Ez bizonyos, időkritikus alkalmazások esetén hátrányos lehet, de a WiFi tipikus felhasználása esetén (mely számítógépek összekötése) nem bír jelentőséggel.

Ahhoz, hogy egy eszköz csatlakozhasson egy már meglévő hálózathoz, először a hálózati paramétereket kell beállítani. A legfontosabb paraméter a használt frekvencia: a WiFi szabvány a 2.4GHz-es tartományban 13 darab 22MHz sáv szélességű, átlapolódó csatornát határoz meg.

A WiFi napjaink meghatározó vezeték nélküli kommunikációs protokolljává vált. Gyakorlatilag minden informatikai eszköz támogatja, a beágyazott rendszerektől a mobiltelefonokon át a számítógépekig. Bár hétköznapi használatra CSMA-alapú közeghozzáférése és nagy adatsebessége miatt ideális, az iparban a nemdeterminisztikus késleltetése és a zajjal szembeni érzékenysége miatt problémákba ütközhet a használata [18].

2.3. Ipari környezetbe szánt kísérleti kommunikációs protokollok

Ahogy arról az előző alfejezetben szó volt, az iparban használt három legelterjedtebb protokoll közül kettő (WiFi és Bluetooth) eredetileg nem ipari körülmények közé lett kifejlesztve, a ZigBee pedig nem elég általános. Ezen hiányosságok miatt több kutató is foglalkozik olyan protokollok kifejlesztésével, amik megfelelnek az ipari vezérlés szükségleteinek. A kísérleti eredmények bemutatása előtt érdemes szót ejteni arról, hogy milyen követelményeknek kell megfelelnie egy ipari WSN-rendszernek.

2.3.1. Az ipari protokollokkal szemben támasztott követelmények

A különböző ipari létesítményekben használt vezeték nélküli protokollok rétegeinek több szigorú követelményt kell teljesíteniük [18]. A fizikai és a MAC réteg tervezésekor fontos, hogy a kifejlesztett kommunikációs protokoll

- **alacsony energiaigényű** legyen, hogy az elemről táplált eszközök élettartama minél hosszabb lehessen;
- **alacsony** legyen az **adóteljesítmény** és **optimális** legyen a **spektrum kihasználtsága**, hogy a hálózat minél kisebb interferenciát okozzon a többi berendezésben;
- **determinisztikus** legyen a **közeghozzáférési réteg**, hogy a válaszidők kiszámíthatóak legyenek.

A hálózati rétegben kifejlesztett protokollnak

- **rugalmasnak kell lennie**, támogatnia kell az olyan alapvető topológiákat, mint a csillag, fa, vagy a mesh-hálózat;
- **skálázhatónak kell lennie**, támogatnia kell a sűrű csomópontok és a nagy (több száz vagy ezer elemet számláló) hálózatok kezelését;
- kezelnie kell a **mozgó csomópontokat**;
- garantálnia kell a kommunikáció **hitelességét és védelmét**.

Ezek a követelmények csak általános útmutatások, a protokoll kiválasztásánál mindig figyelembe kell venni a konkrét alkalmazást is. Egy negyedóránkénti mintavételezéssel mérő környezet-monitorozó rendszer esetében a determinisztikus közeghozzáférés nem olyan fontos, ellenben az alacsony energiaigénnyel. (Ezért használják a ZigBee-t előszeretettel különböző ipari szenzorhálózatok megvalósítására [19][20] annak ellenére, hogy a MAC-rétege nemdeterminisztikus [13].)

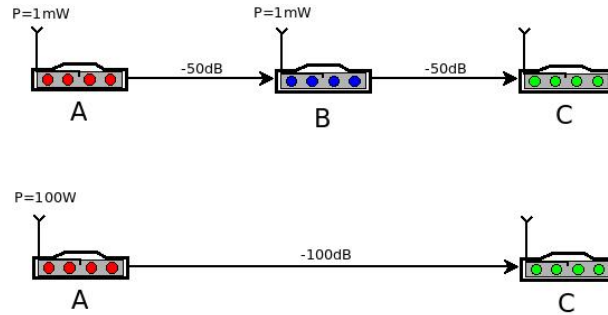
A dolgozat során a cél egy olyan rendszer kifejlesztése, mely képes egy ipari létesítmény berendezéseinek elosztott vezérlésére egy erre alkalmas vezeték nélküli protokoll segítségével. (A vezeték nélküli megoldásnak a vezetékeshez szemben több előnye is van, mint például a csökkentett költségek és a nagyfokú rugalmasság [21].) Mivel a vezérelt eszközök (szelepek, motorok, hőcserélők, ...) energiaigénye nagyságrendekkel nagyobb, mint a csomópontoké, az energiahatékonyság nem olyan fontos tényező (hiszen az állandó energiaellátás az esetek nagy részében könnyen megoldható), viszont a **determinisztikus csomagtovábbítás** (a megbízható vezérlés miatt), az **alacsony késleltetés** (a valós idejű beavatkozás miatt), az **alacsony adóteljesítmény** (néhány berendezés érzékeny lehet az elektromos zajra), a **rugalmasság** (az ipari létesítményekben a topológia nagyon sokféle lehet) és a **skálázhatóság** (előfordulhat, hogy több ezer berendezést kell vezérelni) kiemelt fontosságú tervezési szempont. A protokoll kiválasztásakor fontos azt a tulajdonságot is figyelembe venni, hogy a csomagok többsége **multicast**, ami az alkalmazás következménye: ritka az, hogy egy mért értéket csak egy helyen kell felhasználni. (Egy mért értéket legalább három helyre kell eljuttatni: a beavatkozóhoz, ami a mérési eredmény alapján szabályoz; az operátorhoz és a historikus adatok tárolásáért felelős egységhez.) Hasznos lehet továbbá a mozgó csomópontok támogatása is. A WiFi, ZigBee és Bluetooth nem teljesíti a fenti követelményeket, ezért meg kell vizsgálni a kevésbé elterjedt, kísérleti protokollokat.

2.3.2. A TREE protokoll

A TREE [22] (**R**eal-time **R**outing performance, transmission **R**eliability, and **E**nergy **E**fficiency) tervezése során a cél egy olyan útvonalkeresési stratégia létrehozása volt, mely képes kiszolgálni egy valós idejű, megbízható, energiahatékony protokollt.

A javasolt útvonalkeresési stratégia alapja, hogy a hálózat semmit sem feltételez a csatornáról, minden paramétert és útvonalat működés közben (on-the-fly) határoz meg. Ez a módszer – szemben a matematikai modellekre támaszkodó, optimális útvonalkeresési módszerekkel – nem tartalmazza a modellek hibáit, így egy gyorsan változó, kaotikus környezetben – mely az ipari körülményeket jellemzi – jobban teljesítenek.

Az on-the-fly útvonaltervezéshez meg kell határozni néhány olyan jól mérhető paramétert, melyek hangolásával a hálózat egészének teljesítménye javítható. A TREE tervezése során az egyes útvonalak energiaköltségét és késleltetését hasonlították össze, melyeket a másik rovására lehet növelni (10. ábra.).



10. ábra: Energiahatékonyság $\Leftrightarrow$ Késleltetés tradeoff. Legyen a vevő érzékenysége -50dBm . Ekkor a felső esetben A-nak és B-nek egyaránt 1mW -os adóteljesítménnyel kell adnia. $(-50\text{dBm} + 50\text{dB} = 10 * \log_{10}(\frac{P}{1\text{mW}}) \Rightarrow P = 1\text{mW})$ Amennyiben az átjátszóállomást (B) kihagyva A közvetlenül C-nek üzen (megspórolva ezzel B késleltetését), kétszer akkora veszteséget kell áthidalnia, amihez már 100W -os adóteljesítményre van szüksége. $(-50\text{dBm} + 100\text{dB} = 10 * \log_{10}(\frac{P}{1\text{mW}}) \Rightarrow P = 100\text{W})$ Ez öt nagyságrenddel nagyobb, mint A és B együttes teljesítménye.

Bár az ad-hoc útvonalválasztás jó hatásfokkal működhet, a legnagyobb problémát az okozza, hogy az egyes csomópontoknak úgy kell a hálózat egészét érintő, összehangolt döntéseket hozniuk, hogy csak a közvetlen szomszédokról rendelkeznek információval, ami nem elég az optimális útvonaltervezéshez. A TREE protokoll ezt a problémát UPM üzenetekkel oldja meg. Az UPM (update message) olyan üzenet, amiben a csomópontok kiküldik az összes információt, amivel rendelkeznek a topológiával kapcsolatban. Ez az információ kezdetben csak a közvetlen szomszédokra vonatkozik, de ahogy ezek az üzenetek szétterjednek a hálózaton, a csomópontok képesek összerakni a teljes topológiát.

Ebben az esetben akkor jelentkezhet probléma, ha a hálózat mérete nagyobb annál, mint amit egy csomópont tárolni tud. Éppen ezért az egyes csomópontok nem a hálózat egészéről, csak a tőlük néhány ugrásra lévő csomópontokról tárolnak el az információt. Azzal, hogy minden csomópont a saját térképe alapján optimális útvonalat jelöl ki a csomagnak, jól közelíthető a globális optimum.

Bár a protokoll kifejlesztésekor a cél annak ipari használata volt, a dolgozat során bemutatott alkalmazás kiszolgálására nem alkalmas, mert sem a multicast csomagokat, sem a mozgó csomópontokat nem kezeli hatékonyan.

2.3.3. A SPEED protokoll

A SPEED [23] protokoll tervezése során a cél egy olyan valós idejű protokoll kifejlesztése volt, mely szenzorhálózatok alapjául szolgálhat. A protokoll három különböző címzési módot támogat:

- **unicast címzés** esetén a csomagot egy meghatározott csomópont kapja meg, amit ID alapján lehet azonosítani;
- **terület-alapú multicast** esetén a küldött csomagokat egy adott földrajzi koordináta közelében lévő minden csomópont megkapja. Ezzel a funkcióval el lehet például érni, hogy egy veszélyzóna közelében minden figyelmeztető lámpa bekapcsolódjon. A célba juttatás úgy történik, hogy amint megkapja egy, a földrajzi tartományon belüli csomópont, megfelelő TTL-lel elkezd broadcast-olni azt, így a tartományon belül minden megkapja;
- **terület-alapú anycast** esetén egy adott terület néhány csomópontja kapja meg az üzenetet. Ez szenzorhálózatok esetén azért hasznos, mert az egymáshoz közeli csomópontok mérési értékei korrelálnak egymással. Ha az adatgyűjtő kíváncsi egy szoba hőmérsékletére, felesleges a szobában lévő összes csomóponttól kérnie az információt, mert azok úgy sem fognak érdemben különbözni egymástól. A multicast és anycast csomagok továbbítása hasonlóan történik a kérdéses terület határáig, de ha egy céltartományon belüli csomópont fogad egy terület-alapú anycast csomagot, akkor az már nem továbbítja.

Bár a TREE-protokoll valós idejű és támogatja a multicastnak egy speciális formáját, ez nem alkalmas a dolgozat céljával kitűzött alkalmazáshoz, mert a szükséges multicast csomagok címzettjei csak ritkán vehetők egy csoportba a földrajzi elhelyezkedésük alapján.

2.3.4. A NADV és a LEACH protokoll

A NADV [24] protokoll a LEACH [25] protokoll továbbfejlesztett változata. A fejlesztés során a cél a WSN-ben részt vevő csomópontok élettartamának növelése volt az aktív és alvó üzemmód közötti váltás adaptív ütemezésén keresztül.

Mindkét protokoll alapja a dinamikus csoportokra bontás (klaszterezés). A protokoll a működését körökre osztja. Minden kör elején kijelöl bizonyos számú fejegységet (cluster head). A fejegységekhez csatlakoznak a hozzájuk közel található végpontok, amihez a protokoll ad egy ütemezést is. A csatlakozott végpontok mindig csak a saját fejegységüknek küldenek üzenetet, amik rendezik és tömörítik a befolyt információkat, majd továbbítják azt a nyelőnek. Mivel a fejegységeknek sokat kell kommunikálniuk, az energiafogyasztásuk sokkal nagyobb lesz, mint a végpontoknak, ezért a hálózat élettartamának szempontjából fontos, hogy igazságosan legyenek kiválasztva és minden csomópont sorra kerüljön.

Bár ezek a protokollok nagyon hasznosak akkor, ha több, energiakorlátos szenzornak kell adatokat küldenie egy központi nyelőbe, a dolgozat során bemutatott alkalmazás kiszolgálására alkalmatlan. Ezek a protokollok nem valós idejűek, továbbá nem determinisztikusak és a topológia szempontjából sem elég rugalmasak.

3. Új, elosztott ipari vezérlést megvalósító vezeték nélküli ke-retrendszer

Bár a jelenleg elterjedt központi vezérlés fejlesztése egyszerű és költségei alacsonyak, két hátránya van:

- i.) A valódi döntéshozó logika és a beavatkozás elválnak egymástól. (A döntéseket a PLC hozza meg, az aktuátorokat és szenzorokat viszont egy távoli IO egység kezeli.) Ez érzékennyé teszi a rendszert a hálózat meghibásodására. Sokkal hatékonyabb és hibatűrőbb megoldást eredményez, ha az adatok elemzése a méréshez és a beavatkozáshoz közel történik [26].
- ii.) Az egymástól nagyrészt független részegységeket is ugyanaz a vezérlő irányítja, tehát az egyik rendszer karbantartása érintheti a többi rendszert is. (Akár azok felesleges leállását okozva.)

Jelen dolgozat során a cél egy olyan rendszer lefejlesztése, ami a döntéshozó logikát át-helyezi a tényleges beavatkozás helyére anélkül, hogy a fejlesztést megnehezítené vagy a költségeket növelné.

A dolgozat során megtervezett rendszer (**TinyPLC**) logikailag három részre osztható:

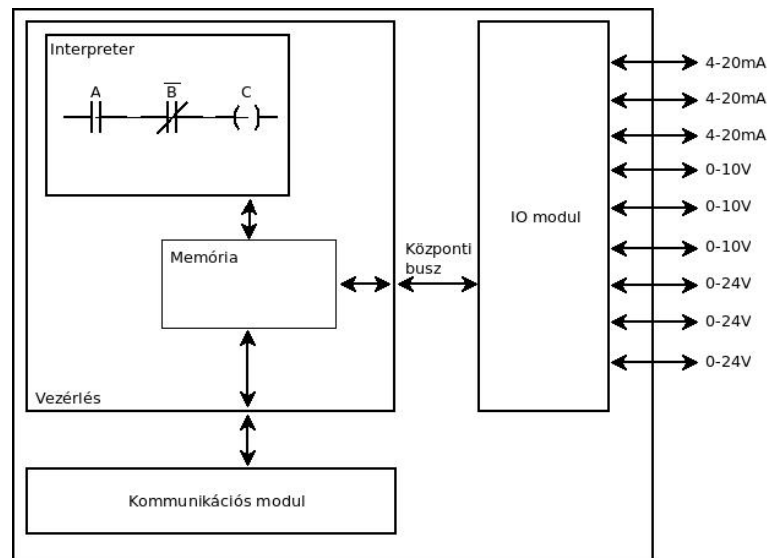
- i.) A rendszer alapját a **végpontok** alkotják, melyek képesek futtatni a fejlesztők által elkészített programot. A teljes vezérlés több száz végpontot is tartalmazhat, ezért ezek költségét alacsonyan kell tartani, de ez nem mehet a minőség vagy az általánosság rovására.
- ii.) A végpontokat összekötő **kommunikációs protokoll** gondoskodik arról, hogy az egyes rendszerek képesek legyenek megosztani az állapotukat és reagálni a globális hibákra. Sok olyan szituáció fordulhat elő, amikor – bár az alrendszerek egymástól függetlenül működnek – az egyik részegység leállása vagy hibája befolyásolja a teljes gyártást. (Például egy szivattyúcsoporthoz nem lehet addig elindítani, amíg a vízel-látásért felelős alrendszer nem biztosítja a megfelelő nyomást.) A kommunikációs protokollnak kellően rugalmasnak és robusztusnak kell lennie és biztosítania kell a hálózat monitorozását, valamint a kimaradt csomagok észlelését és lekezelését.
- iii.) Szükség van egy olyan **fejlesztői szoftvercsomagra** is, amin keresztül a fejlesztők beállíthatják, felprogramozhatják és monitorozhatják a rendszer működését. A rendszert több száz, egyenként programozható és konfigurálható modul alkotja, amik között nagy fizikai távolság, így a fejlesztőktől nem várható el, hogy ezeket külön-külön kezeljék. A fejlesztőkörnyezet feladata, hogy az egész projektet össze-fogja és amennyire csak lehet, elrejtse a fejlesztők elől annak széttagságát.

3.1. A végpontok hardver és szoftver komponensei

A végpontok szerkezetileg két nagyobb részegységre oszthatóak, a vezérlésre és a IO részegységre (11. ábra). A vezérlés tárolja és futtatja a firmware-t (ami mindegyik végpont esetén megegyezik) és a fejlesztők által elkészített logikát, valamint biztosítja a kommunikációt.

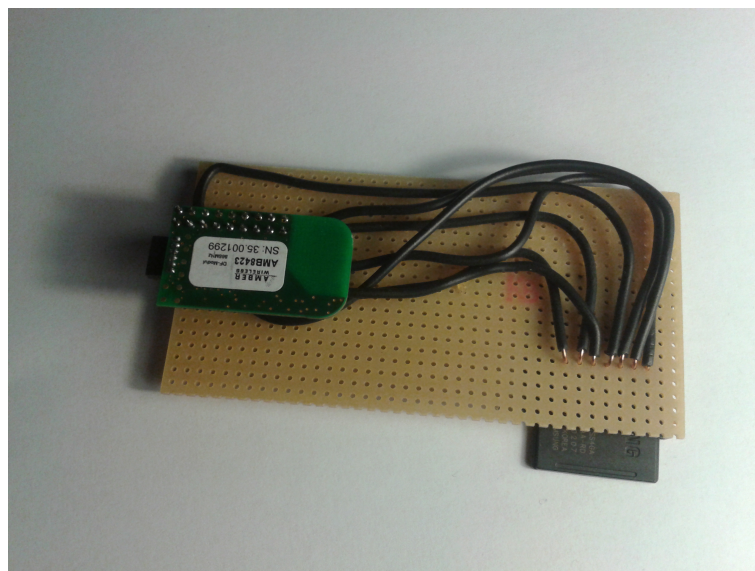
Az IO modul feladata, hogy tartsa a kapcsolatot az aktuátorokkal és a szenzorokkal. Az ipari rendszerek jellemzően 4-20mA-es, 0-10V-os, vagy 0-24V-os jelszinteken működnek,

ami különbözik a prototípusban használt mikrovezérlő jelszintjeitől. A kettő közötti illesztést az IO részegységnek kell biztosítania. Ezen kívül biztosítania kell további IO-pontok illesztését is egy multiplexer segítségével. Az ehhez szükséges áramköröket egyszerű és olcsó legyártani, ezért a dolgozatban nem lesz róla szó.



11. ábra: A modul vázlatos felépítése. A blokkdiagram tartalmazza a vezérlést, az IO modult és a kommunikációs modult.

A végpont irányítását végző részegységnek tartalmaznia kell egy mikrovezérlőt, egy nagykapacitású adattárolót és egy kommunikációs modult. A prototípus magja egy AMB8423-as modul, amihez egy saját áramkörön keresztül csatlakozik az SD-kártya. (12. ábra)



12. ábra: A modul korai prototípusa. Felül az Amber modullal, alul pedig a csatlakoztatott SD-kártya látható.

Az AMB8423-as modul tartalmaz egy MSP430F2274 típusú mikrovezérlőt és egy CC1101-es rádiómodult, így ideális prototípusa egy ilyen jellegű rendszernek. Az MSP430F2274-es mikrovezérlő C nyelven programozható, rendelkezik ki- és bemeneti portokkal, valamint ADC-vel is, bár csak 0-3.2V-os jelszintekkel tud dolgozni. (A 0-3.2V és az ipari jelszintek közötti váltás az IO modul feladata.) Rendelkezik SPI interfésszel is, amin keresztül képes

kommunikálni az SD-kártyával, illetve írni és olvasni annak tartalmát.

A CC1101-es rádiómodul rugalmas, könnyen konfigurálható, a segítségével tetszőleges protokollt lehet implementálni.

3.1.1. A végpontok programozása

A végpontok tartalmazzanak egy mikrovezérlőt, ami C nyelven programozható, de ezt a végfelhasználók elől el kell rejteni. A C nyelv több okból sem megfelelő választás arra, hogy ipari körülmények között elkészítsék benne a vezérlést:

- a C nyelvű vezérlőkód **nem biztonságos**: mivel a fejlesztő a mikrovezérlő minden funkcióját elérheti, nem lehet garantálni, hogy egy programozási hiba nem akadályozza a firmware-t valamilyen funkcióban, ellehetetlenítve ezzel a végpont működését;
- a C nyelvű programozás **sok hibalehetőséget rejt magában**: például a memória helytelen kezelése, ezért alapos tesztelést igényel;
- az **IEC 61131-3 szabvány nem tartalmazza** a C nyelvet.

A fentiek miatt biztosítani kell a modulok felprogramozásához egy egyszerű, szabványos nyelvet, amit az előre beégetett firmware felügyelten futtathat. A most bemutatásra kerülő rendszerben ez a létradiagram egy bináris prezentációja, ami az eredeti diagram alapján könnyen és egyértelműen generálható. (A generálás algoritmusáról a 3.3. fejezetben lesz szó.)

A 3. és a 2 táblázatok a jelenleg támogatott utasításkészletet és az utasítások lehetséges címzési üzemmódjait tartalmazzák. (A címzési üzemmódokról és a memóriakezelésről a 3.1.2. alfejezetben lesz szó.) Kétféle utasítás különböztethető meg, az első csoport az program vezérléséért felelős, a második pedig az adatok kezeléséért.

2. táblázat: A programvezérlő utasítások.

Utasítás neve	Jelentése
BST	Elágazás kezdődik a létrában
NXB	Új elágazás egy már létező BST-től
BND	Elágazás lezárása
CAL	Ugrás a programmemória adott pontjára, ha a rung condition igaz.
EOR	Rung lezárása. (A rung condition igaz lesz.)
EOP	Program vége. Ha a mostani programra egy CAL hatására került a vezérlés, akkor a vezérlés onnan folytatódik, egyébként véget ér az utasításciklus

3. táblázat: Az adatkezelő utasítások. Felépítésük nagyon hasonló egymáshoz. Például a MOV A,B hatására A értéke (ami lehet konstans vagy memóriahivatkozás) beemásolódik B helyére (ami kötelezően egy memóriahivatkozás).

Utasítás neve	Utasítás jelentése	C	D	I
XIC A,b	Ha a bemenete igaz és A igaz, a kimenete is igaz	x	x	x
XIO A,b	Ha a bemenete igaz és A hamis, a kimenete is igaz	x	x	x
OTE A,b	Ha a bemenete igaz A igaz lesz, egyébként hamis.		x	x
OTL A,b	Ha a bemenete igaz A igaz lesz. (Latch)		x	x
OTU A,b	Ha a bemenete igaz A hamis lesz. (Unlatch)		x	x
SWP A,B	A és B értékét felcseréli		x	x
MOV A,B	B:=A	x(1)	x	x
ADD A,B,C	C:=A+B	x(1,2)	x	x
SUB A,B,C	C:=A-B	x(1,2)	x	x
MUL A,B,C	C:=A*B	x(1,2)	x	x
DIV A,B,C	C:=A/B maradék nélkül	x(1,2)	x	x
INC A	A:=A+1		x	x
DEC A	A:=A-1		x	x
EQV A,B	Ha A==B és a bemenet igaz, akkor a kimenet igaz	x	x	x
NEQ A,B	Ha A<>B és a bemenet igaz, akkor a kimenet igaz	x	x	x
LES A,B	Ha A<B és a bemenet igaz, akkor a kimenet igaz	x	x	x
GRT A,B	Ha A>B és a bemenet igaz, akkor a kimenet igaz	x	x	x
LEQ A,B	Ha A<=B és a bemenet igaz, akkor a kimenet igaz	x	x	x

A firmware felépítése moduláris, könnyű új utasításokkal bővíteni.

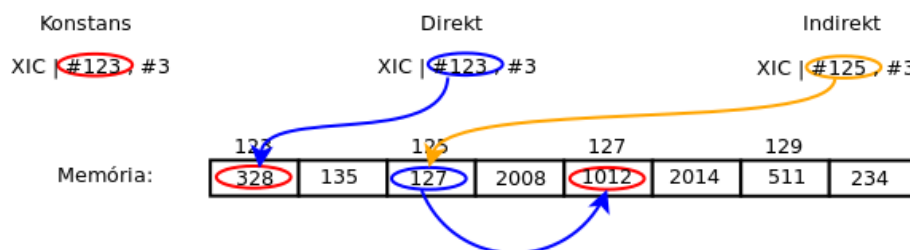
3.1.2. Memóriakezelés

A végfelhasználók szempontjából a firmware memóriakezelése Harvard architektúra [27] szerint épül fel, a programmemóriához pedig csak korlátozott hozzáférést kapnak.

Adatokat három módon lehet olvasni:

- a **konstansok** a programmemóriában tárolódnak, közvetlenül az utasítások után (ez csak olvasható lehet);
- az utasítások után állhat egy **direkt memóriacím** is, ekkor a megadott memóriaterületen történik az írás és az olvasás;
- a harmadik címezési mód az **indirekt címezés**, ekkor az utasítás után szereplő címről olvassa ki a firmware azt a memóriacímet, amin az utasítás dolgozik.

A különböző memóriahozzáférési módokat mutatja be a 13. ábra.



13. ábra: A prototípus interpreter **memóriakezelése**. Konstans esetén a felhasznált érték 123, direkt elérés esetén 328, indirekt esetén pedig 1012.

Azt, hogy egy utasítás milyen típusú címezési módokat használ, a fejlesztőkörnyezet dönti el. A firmware memóriakezelésének részletes leírása a 3.1.3. fejezetben található.

A fejlesztők természetesen nem csak belső memóriacímeken akarnak dolgozni, hanem más csomópontoktól kapott adatokon, vagy fizikai ki- és bemeneteken. A firmware úgy

rejt el az egyes adatforrások közötti különbséget, hogy az utasításciklus elején minden bemenetet a memória egy meghatározott részébe ír, a ciklus befejeztével pedig minden kimenetet a memóriából kiír a megfelelő helyre, a végfelhasználó elől ez elfedésre kerül. (A 3.1.3. fejezetben ennek a funkciónak a működése is részletesen ismertetésre kerül.)

3.1.3. A firmware bemutatása

A firmware feladata, hogy futtassa és felügyelje a végfelhasználó által írt programot, elrejtse a különböző adatforrások közötti kommunikációt és kezelje a perifériákat. C nyelven történt az implementációja oly módon, hogy a platformfüggetlen és platformfüggő kódreszletek élesen elkülönüljenek egymástól, egyszerűsítve ezzel egy esetleges új architektúrára történő átállást.

A program belépési pontja a main-függvény, melyben a részegységek inicializációja után egyből megkezdődik az utasításciklus. Az utasításciklus három részre osztható:

- i.) először a **bemenetek** memóriába történő beolvasása történik meg;
- ii.) ezután lefut a végfelhasználó által írt **vezérlőprogram**;
- iii.) végül a **kimenetek** a memóriából a tényleges kimeneti tárba kerülnek.

A második lépésben történik meg a memóriában és az SD-kártyán tárolt program beolvasása és futtatása. (A memóriakezelő függvények elrejtik a kettő forrás közötti különbséget.) A programmemória (az adatmemóriához hasonlóan) word-önként érhető el, és az utasítások mérete a paraméterek számával változik.

Az interpreter működése

Mivel a mikrovezérlő 16 bites, ezért az interpreter is 16 biten ábrázolja a számokat, 16 bites memória- és kódszavakat használ. A memóriaelérésért felelős függvényeket a *memory.h* és *memory.c* tartalmazza, amiben az alábbi függvények vannak implementálva:

- **void** *writeDataMem*(dataAddress addr, dataWord data, dataWord bitmask);
A függvény a data adatot az addr címre írja, de csak azok a bitek változnak, amik a bitmask-ban 1-es értékűek.
- dataWord *readDataMem*(dataAddress addr);
Visszatér a memória értékével az adott címen
- **void** *setDataBit*(dataAddress addr, bitAddress bit, bitData data);
Az adott címen 1-re állítja a bit-edik bitet ha data igaz, egyébként nullázza.
- bitData *getDataBit*(dataAddress addr, bitAddress bit);
Visszaadja a kért bit értékét
- codeWord *getCodeMem*(codeAddress addr);
Visszaadja a kódmemória adott címén lévő értéket
- **void** *setCodeMem*(codeAddress addr, codeWord data);
Átállítja a kódmemória adott címén lévő értéket

A *PLC.c* tartalmazza az interpreter kódját, ami a memóriát a *memory.h* függvényein keresztül éri el. Egyetlen függvénye, a *run(char rc, codeAddress *i)*, melynek feladata, hogy a *codeAddress*-től kezdve olvassa a kódmemóriát és végrehajtsa az ott található utasításokat.

Minden utasítás három részre osztható:

- i.) Az első 8 bit az első négy paraméter címezési módját határozza meg, melyek közül az első két bit az első paraméterét a második bitpár a második paraméterét, és így tovább. A bitpárok jelentése a következő:
 - 00 : Konstans
 - 01 : Nem használt
 - 10 : Direkt elérés
 - 11 : Indirekt elérés
- ii.) A második 8 bit az utasítás kódját tartalmazza, amiből következik, hogy összesen 256 beépített utasítás lehetséges e szerint a szisztéma szerint. Ez a korlát enyhíthető a függvényhívás bevezetésével.
- iii.) Az utasítás további része a paraméterlista. Mivel az első 8 bit csak négy paraméter elérési módját határozta meg, ezért négy paraméter után be kell vezetni egy új, 16 bites szót, ami a következő (immár) nyolc paraméter címezési módját határozza meg.

Az interpreter szekvenciálisan beolvassa a programmemória tartalmát és végrehajtja az ott található utasításokat. Amennyiben elágazáshoz ér (*BST* és *NXB* utasítások), rekurzívan meghívja önmagát és úgy folytatja a kiértékelést.

Példa az interpreter használatára

Az interpreter használatát jól lehet szemléltetni egy buborékrendezésen keresztül. Először az adatmemória és a kódmemória tartalmát kell meghatározni:

```

1 dataWord dataMemory []={4,11,0,0,43,46,11,45,27,34,21,67,23};
2
3 codeWord codeMemory []=
4 {BST, BIT15+BIT13+LES, 2, 0, NXB, BIT15+BIT13+GRT, 2, 1, BND, BIT15+BIT13+
   MOV, 0, 2, EOR,
5 BIT15+BIT11+ADD, 2, 1, 3, EOR,
6 BIT15+BIT14+BIT13+BIT12+LES, 2, 3, BIT15+BIT14+BIT13+BIT12+SWP, 2, 3, EOR,
7 BIT15+INC, 2, EOR,
8 EOP};

```

Az adatmemória első négy szavát a program használni fogja a futáshoz, az utána következőket pedig rendezi. (Az első két szó határozza meg, hogy mely részek legyenek rendezve, ez a 4 és 11 közötti címen található elemek.)

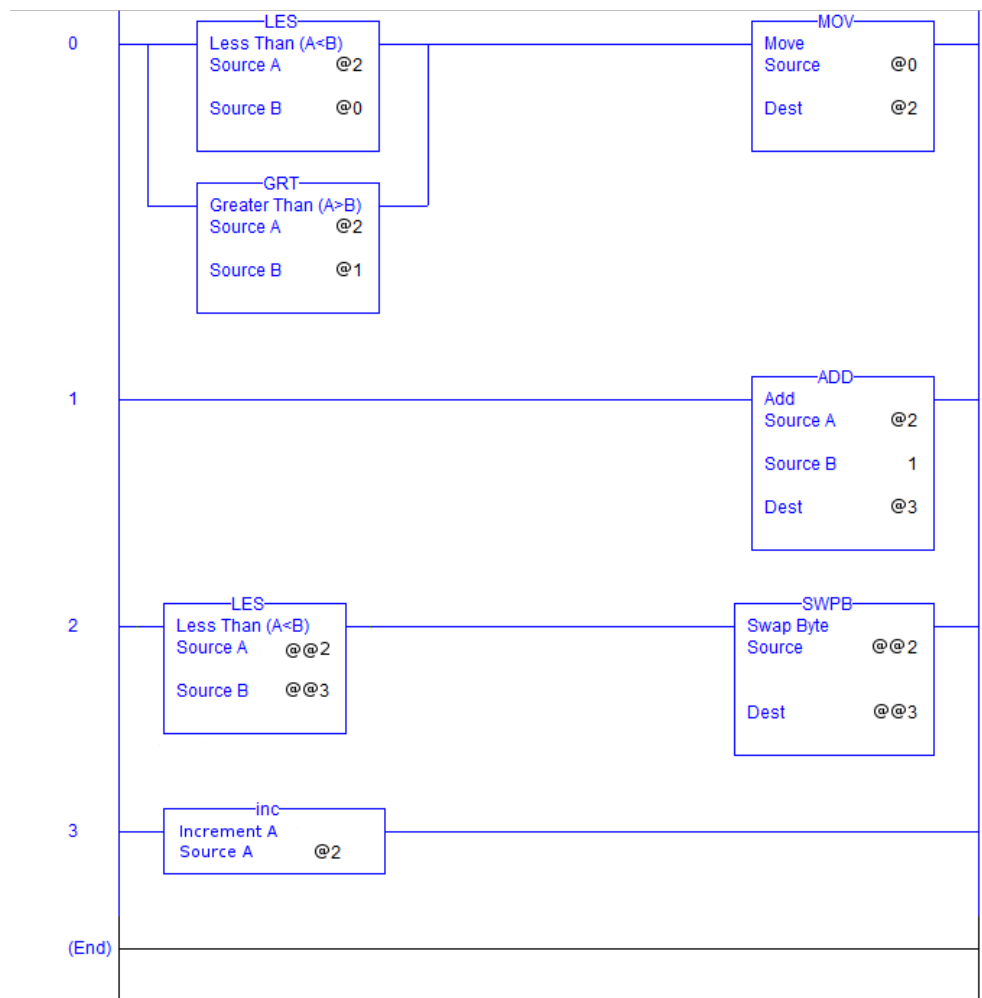
A programmemória valójában egy létrakód szöveges prezentációját tartalmazza, melynek a grafikus megfelelője a 14. ábrán látható. A címezési módokat a @ jel jelöli a létrakódban, az alábbiak szerint értelmezve:

- a @ nélküli szám **konstanst** jelent;
- a @NNN alakú szám **direkt címezést** jelent;

– a @@@NNN alakú szám **indirekt címzést** jelent.

A 0. rung szerepe, hogy gondoskodjon arról, hogy a @2 ne kerülhessen ki a rendezési címtartományból. A @0 helyen a címtartomány alja, @1 helyen a címtartomány teteje előtti érték van. Az 1. rung a @3 mutatót állítja be, hogy mindig @2 után mutasson. A 2. rungban a buborékrende­zés cseréje látható. Amennyiben A @2 által hivatkozott érték kisebb, mint az utána következő, akkor a két számot megcseréli. Itt látható példa az indirekt címzésre. (Ez a rendező algoritmus csökkenő sorrendbe rendez.) Az utolsó rungban a @2 léptetése történik meg.

Mivel a 4 rung ciklikusan fut, ezért a tömb hosszának négyzetének megfelelő számú beolvasási ciklus után a tömb rendezett lesz.



14. ábra: Buborékrende­zést megvalósító létrakód. A 0. rung szerepe, hogy gondoskodjon arról, hogy a @2 ne kerülhessen ki a rendezési címtartományból. A @0 helyen a címtartomány alja, @1 helyen a címtartomány teteje előtti érték van. Az 1. rung a @3 mutatót állítja be, hogy mindig @2 után mutasson. A 2. rungban a buborékrende­zés cseréje látható. Amennyiben A @2 által hivatkozott érték kisebb, mint az utána következő, akkor a két számot megcseréli. Itt látható példa az indirekt címzésre. (Ez a rendező algoritmus csökkenő sorrendbe rendez.) Az utolsó rungban a @2 léptetése történik meg.

3.2. A kommunikációs protokoll

A kommunikációs protokoll algoritmikusan helyes és hatékony megvalósítása a rendszer működésének a szempontjából kritikus, hiszen több száz csomópont között kell azt stabilan biztosítani. A kommunikációs protokollokat jellemzően az OSI-modell alapján egymástól független rétegekre bontva szokták meghatározni, de ez egy WSN esetén nem ad feltétlenül optimális eredményt. A dolgozat éppen ezért törekszik az OSI-modell szerinti tagolás megtartására, de az egyes rétegek a Cross Layer Design[28] útmutatásai szerint egymással több ponton is összekapcsolódhatnak, ami sértheti az OSI-modellt.

A csomópontok közötti kommunikációnak több követelménynek kell egyidejűleg megfelelnie:

- **Folyamatosan küldött, multicast-jellegű csomagok:**

Amennyiben egy paraméterre (ami lehet akár egy mért érték is, vagy egy státuszinformáció) több csomópontnak is szüksége van, akkor azt valahogy meg kell osztani az érintettek között. Mivel nem biztos, hogy az adott paraméter minden felhasználója szomszéd, ezért az átvitel során szükség van közbeeső továbbítókra is. Ezeknek a csomagoknak nincs konkrét címzettjük, minden olyan csomópont fel fogja őket használni, aminek szüksége van (akár csak továbbítás céljából) a csomagban lévő egyik paraméterre. Egy adott paraméterre feliratkozott végpontok consumer-ek (fogyasztók), míg a paraméter forrása (például a konkrét mérést végző csomópont) a producer (előállító).

- **Globális konfigurációs üzenetek szétosztása:**

A hálózat beállításakor, valamint információk lekérdezésekor szükség van arra, hogy egy konfigurációs üzenet minden csomóponthoz eljusson. Ilyenkor úgy kell bejárni a teljes hálózatot, hogy nem szabad semmit feltételezni annak topológiájáról. (Kivéve azt, hogy összefüggő gráfot alkot.) A protokoll felsőbb rétegeinek ismertetésekor látható lesz, hogy ezek a csomagok unicast-címzéssel valósíthatóak meg.

- **Célzott konfigurációs üzenetek célba juttatása:**

Egy modul konfigurálásakor és monitorozásakor olyan konfigurációs üzeneteket kell eljuttatni két csomópont között, amik nem tartoznak a többi csomópontra. A hálózat topológiája ekkor már ismert, a feladat az, hogy a csomag minél rövidebb úton célba jusson. Erről is látható lesz, hogy unicast üzenetek sorozatával lehet a legkönnyebben megvalósítani.

A követelmények mellett meg kell vizsgálni, hogy a feladatból kifolyólag milyen egyszerűsítéseket lehet tenni az általános esethez képest. A két legfontosabb feltétel, hogy a topológia nagyon ritkán változik, valamint a csomópontok közötti adatforgalom előre meghatározható. (Hiszen a végpontok mindig ugyan azokat a paramétereket használják.)

3.2.1. Fizikai réteg

A fizikai réteg feladata, hogy két csomópont között lehetővé tegye a kommunikációt. Erre számtalan megoldás ismert, jelen rendszer prototípusa a CC1101-es rádiómodul lehetőségeit használja ki, ami a 779-928MHz-es frekvenciatartományban működik.

A modul többféle modulációs sémát és kommunikációs üzemmódot biztosít, szabadon konfigurálható. Jelen dolgozat során a fizikai réteg nem játszik jelentős szerepet, bár később érdemes lehet megvizsgálni, hogy a paraméterek adaptív hangolásával hogyan lehet növelni a kommunikációs linkek megbízhatóságát.

3.2.2. Adatkapcsolati réteg

Az adatkapcsolati réteg feladata, hogy biztosítsa két szomszédos csomópont közötti megbízható adatátvitelt. Ez további két alrétetre osztható: Az LLC-re (Logical Link Control), ami a hibamentes átvitelt garantálja és a MAC-re (Media Access Control), ami a címezést és a közeghozzáférést vezérli.

Az LLC réteg két feladata a hibajavítás és a keretek átvitelének biztosítása. (Ez utóbbi általában visszaigazolás segítségével éri el.) A most bemutatott protokollban három, funkcióiban és tulajdonságaiban különböző csomagfajta átvitelét kell megoldani. Ezek közül egy multicast, kettő pedig unicast. Mindháromban közös, hogy a csomag hibamentes átvitele megfelelő kódolással vagy a csomag után illesztett CRC-kóddal könnyen megállapítható. Ezt a CC1101-es rádiós modul elvégzi, ezért erről a továbbiakban nem lesz szó. Az átvitel biztosítása a unicast csomagok esetén szintén megoldottnak tekinthető a klasszikusan használt visszaigazolás (acknowledgement) segítségével. (Bár a visszaigazolás pontos mikéntjén lehet optimalizálni, gyorsabb csomagátviteli sebességet érve el ezzel, de ehhez először a felsőbb rétegeket kell tisztázni.)

A multicast üzenetek küldése esetén a visszaigazolás több okból is problémás. Egyrészt a címzetteket nem feltétlenül ismeri a feladó, másrészt a több csomóponttól érkező visszaigazolásokat is ütemezni kell, különben összekeveredhetnek egymással. Ez túlságosan bonyolítaná a protokollt és csökkentené annak átviteli sebességét, ezért érdemes más megoldást keresni, ami az alkalmazásból következik. A felsőbb rétegek tárgyalásakor arról is lesz szó, hogy a multicast csomagokat rövid időközönként periodikusan kell küldeni, tehát egy-egy csomag elvesztése nem okoz gondot az alkalmazás egészére nézve. Azt viszont le kell kezelni a felsőbb rétegekben, ha egy paraméter frissebb értéke (aminek valamelyik multicast-csomaggal kellett volna érkeznie) már régóta nem érkezett meg.

A MAC réteg két feladata a címezés és a közeghozzáférés kezelése. Mivel az alkalmazás során a csomagok nagy része periodikusan ismétlődik, ezért a TDMA (Time division multiple access) kézenfekvő megoldásnak tűnik. Itt két kérdés merül fel:

- i.) Egyrészt a TDMA során minden csomópontnak van egy meghatározott időrése, amelyben csak neki van lehetősége üzeneteket adni. Ez indokolt akkor, ha a hálózatot reprezentáló gráf teljesen összekötött, de a dolgozat során bemutatott alkalmazásban ez nagyon ritka. A 3.2.3. fejezetben bemutatásra kerül egy ütemező, ami a hálózat topológiájának és a forgalmának ismeretében képes úgy kiosztani az időreszeket, hogy egy időben a lehető legtöbb csomópont adhasson, javítva ezzel a hálózat áteresztőképességét.
- ii.) A másik probléma a TDMA-val az, hogy valahogy le kell kezelnie a nem ütemezett unicast-csomagokat is. Erre egy lehetséges megoldás az, ha fenntart egy időrest csak a unicast-üzenetek számára, így azok biztosan nem fogják megzavarni az ütemezett kommunikációt. (Hasonló megoldást alkalmaz a ControlNet is, ahol a kommunikációs busz ciklus egyik felében az ütemezett kommunikáció történik, a másik felében pedig engedélyezett az aciklikus kommunikáció [29].)

A címezési módot is unicast és multicast esetekre kell bontani. Unicast esetében szükség van egy konkrét címre, ami alapján a feladó meg tudja határozni a címzettet. Multicast keretnél a feladó csak beleteszi a keretbe az információkat, amiről tudja, hogy egy-vagy több szomszédjának szüksége van rá. Az összes szomszéd veszi ezt a csomagot, feldolgozza és ha talál benne olyan adatot, amire szüksége van, akkor azt kiolvassa belőle. Ebben az esetben nem szükséges, hogy a csomag tartalmazzon címet.

3.2.3. Hálózati réteg

A hálózati réteg feladata a hálózaton belüli forgalomirányítás. Jelen alkalmazás esetében ez három részre osztható:

- i.) el kell juttatnia a hálózati konfigurációs üzeneteket az összes csomóponthoz, ismeretlen topológia mellett;
- ii.) el kell juttatnia a célzott konfigurációs üzeneteket a címzetthez, ismert topológia mellett;
- iii.) gondoskodnia kell arról, hogy a megosztott paramétereket minden consumer megkapja, ismert topológia mellett.

A célzott konfigurációs üzenetek használhatóak arra, hogy a fejlesztő távolról felprogramozza vagy monitorozza a végpontokat. Célba juttatásuk egyszerű, hiszen a topológia ismert, tehát a feladó meg tudja határozni a teljes útvonalat és el tudja tárolni azt a csomagban. Erről a dolgot további részében nem is lesz szó. A hálózati konfigurációs üzenetek célba juttatása ezzel szemben több okból is bonyolultabb, amiről a 3.2.3. alfejezetben külön lesz szó.

A megosztott paraméterek küldése és fogadása az alkalmazás alapját képezik, itt fontos az optimalizáció, a megbízhatóság és a monitorozhatóság. Ezek ismertetése a 3.2.3. és a 3.2.3. alfejezetben történik meg.

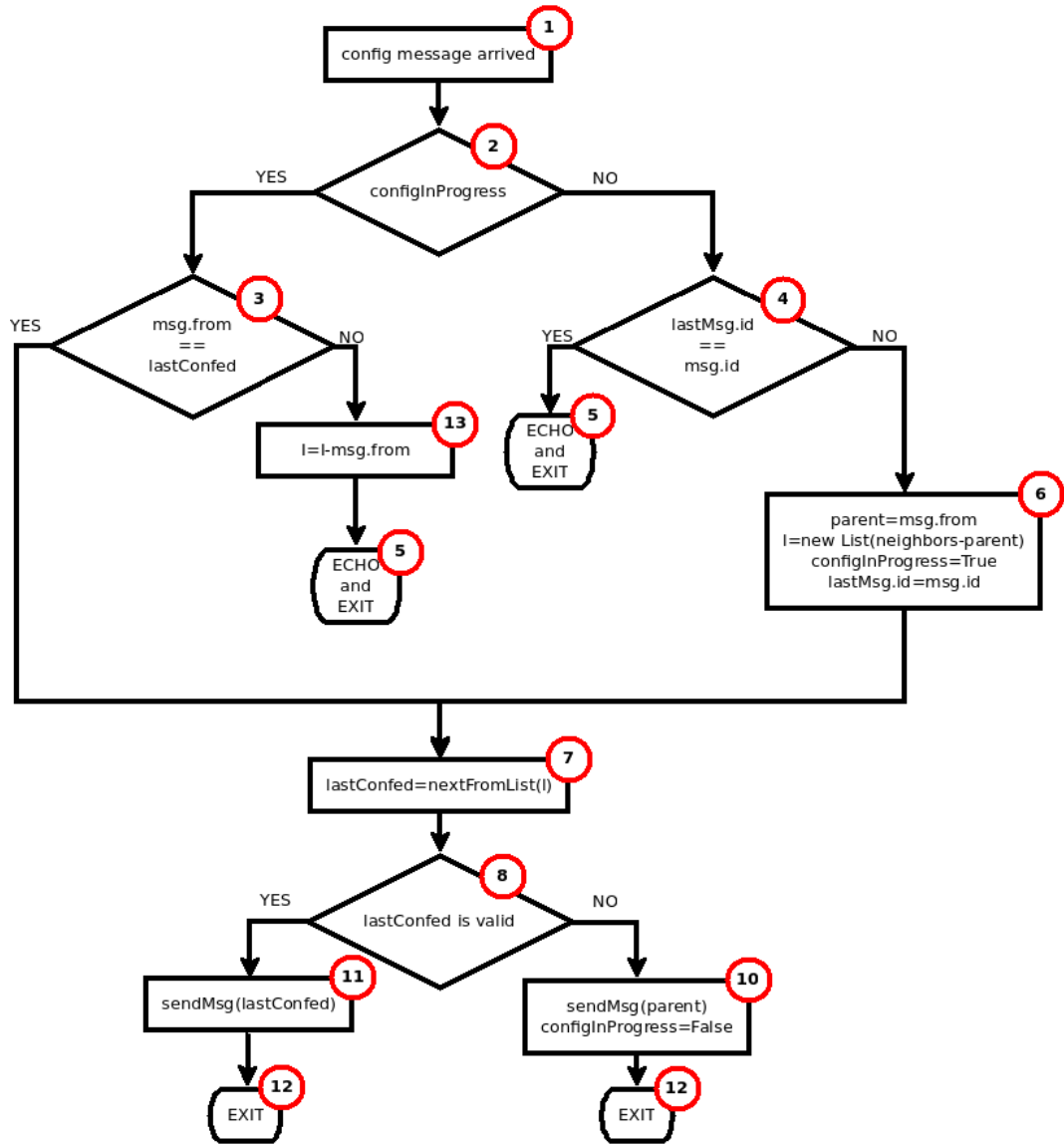
Globális konfigurációs üzenetek

A globális konfigurációs üzenetek célja a hálózat felderítése és paramétereinek beállítása (időrések száma és kiosztása). Ahhoz, hogy ezek az üzenetek akadály nélkül haladhassanak a hálózaton, szükség van egy külön, konfigurációra fenntartott időrésre, ahogy arról a 3.2.2. fejezetben már volt szó. Ebből az is következik, hogy nem lehet egyszerre kettő konfigurációs üzenet a hálózatban, mert azok ütközhetnek.

A globális konfigurációs üzeneteknek úgy kell végigmenniük az egész hálózaton, hogy annak topológiája ismeretlen, az egyes csomópontok csak a saját szomszédait ismerik. Ennek megvalósítására különböző gráfbejárási algoritmusok alkalmazhatóak. A szimuláció (mely az 5.3. fejezetben kerül bemutatásra) a mélységi bejárást használja. Mivel a topológia általában nem körmentes gráf, ezért meg kell akadályozni, hogy az üzenet keringjen a hálózaton. A pontos algoritmus, ami a konfigurációs üzenetek kezeléséért felel, a 15. ábrán látható, a felhasznált változók magyarázattal együtt pedig a 3.2.3. táblázatban.

Változó neve	Változó szerepe
configInProgress	Igaz, ha a modul konfigurálja a szomszédait.
msg.from	Az üzenet közvetlen feladójának ID-jét tartalmazó része
lastConfed	A konfigurálandó szomszéd ID-je
msg.id	A konfigurációs folyamat egyedi azonosítója
lastMsg.id	A legutóbbi konfigurációs folyamat egyedi azonosítója
parent	A konfigurációt kezdeményező csomópont ID-je.
I	A konfigurálandó szomszédokat tartalmazó lista

Az algoritmus végrehajtása akkor kezdődik **(1)**, amikor a modul egy konfigurációs üzenetet fogad. Konfigurálás szempontjából a csomópont két állapotban lehet, amit a *configInProgress* állapotjelző **(2)** tárol. Amennyiben van egy lezáratlan konfigurációs folyamata (vagyis elkezdte konfigurálni a szomszédait és még nem jeleztek neki vissza), az algoritmus ellenőrzi, hogy az üzenet attól a szomszédjától jött-e, akit legutóbb konfigurált **(3)**.



15. ábra: A globális konfigurációs paraméterek az itt látható algoritmus alapján járják be a hálózatot.

Ha nem, akkor a hálózati topológia tartalmaz egy kört. Ilyenkor csak vissza kell küldeni a feladónak az üzenetet (5). A válasz hatására a feladó folytatni fogja a többi szomszédjának a felkonfigurálását. A 13-as lépés nem befolyásolja érdemben az algoritmus működését, de azzal, hogy az **I** listából kiveszi a már biztosan felkonfigurált szomszédot (hiszen egy csomópont csak akkor lehet feladó, ha már maga is kapott konfigurációs üzenetet), csökkenti az elküldendő üzenetek számát.

Ha az üzenet a legutóbb konfigurált csomóponttól származik, akkor az azt jelenti, hogy a hozzá tartozó részgráf konfigurációja befejeződött. Ekkor az **I** listából ki kell választani a következő szomszédot (7). Ha a lista tartalmaz még elemeket, akkor a **lastConfed** érvényes lesz (8) és el lehet neki küldeni a konfigurációs üzenetet. Ha az **I** lista már üres, akkor az azt jelenti, hogy az összes csomópont konfigurációja megtörtént, ezt jelezni kell a szülőnek és ki kell lépni a konfigurációs módból (10).

Visszatérve a 2-es elágazáshoz, amennyiben a csomópont nincs konfigurációs üzemmódban, az két okból lehet (4). Vagy már megtörtént a konfigurációja ebben a konfigurációs folyamatban, ekkor csak vissza kell küldenie a kapott üzenetet (5), vagy még tényleg

nem kezdődött meg. Ekkor inicializálnia kell a **parent** (a szülő az a csomópont, amelyik a konfigurálást kezdeményezi) és **I** (a szülőn kívül minden szomszédot konfigurálni kell) változókat, a **lastMsg.id** értékét be kell állítania és be kell lépnie a konfigurációs üzemmódba **(6)**. (Bár a folyamatábrán erről nincs szó, ez az a lépés, ahol a konfigurációs üzenetben küldött egyéb paraméterek feldolgozása megtörténik.) Ezután az algoritmus végrehajtása a **7-es** ponttal folytatódik, amiről már volt szó.

Fontos, hogy a hálózati csomópontok paraméterei addig nem változhatnak, amíg mind-egyikhez el nem jutott a konfigurációs üzenet. Ellenkező esetben előfordulhat, hogy az egyik csomópont az eredeti konfigurációs időszelvényben próbálja küldeni a konfigurációt, egy szomszédja viszont már az új felosztásban ugyan akkor szeretne küldeni egy másik típusú üzenetet, interferenciát okozva ezzel a hálózatban, ellehetetlenítve a konfigurációs üzenet továbbítását. Ezt a legegyszerűbben várakoztatással lehet megoldani.

A 15. ábrán látható algoritmus csak arra ad választ, hogy hogyan lehet a topológia ismerete nélkül eljuttatni egy csomagot minden végpontnak, de a topológia felderítésére nem. Ehhez bővíteni kell úgy, hogy a szülőnek minden csomópont visszaküldi a szomszédait és a gyerekeitől kapott adatokat is. Így a topológiát leíró információ visszaáramlik a feszítőfa gyökerébe.

Producer-consumer kommunikációs protokoll

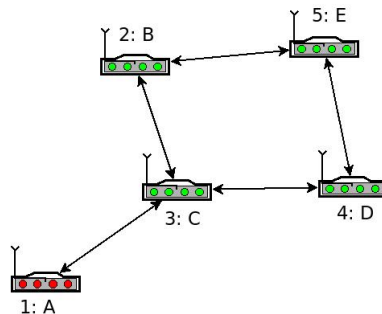
Ahogy arról már volt szó, a csomópontok a paraméterek megosztásának a szempontjából producerekre és consumerekre oszthatóak. Minden paraméterhez tartozik egy producer végpont (amelyik a tényleges mérést vagy számítást elvégzi) és meghatározatlan számú consumer, amik ezt az eredményt felhasználják.

A protokoll alapötlete, hogy a csomópontok egy multicast-üzenetben elküldik a szomszédaiknak a náluk lévő megosztott paramétereket függetlenül attól, hogy producere az adott paraméternek, vagy sem. A csomagokat elkapó szomszédok ezután a náluk lévő értéket felülírják az újonnan kapottal, így a producertől előbb-utóbb el fog jutni az újonnan mért érték az összes consumer-ekhez.

A tervezés során két problémát kellett megoldani. Az egyik az, hogy semmi sem garantálja, hogy a producer és a consumer-ek egy összefüggő gráfot alkotnak. Ekkor szükség van olyan közbeeső végpontokra, amiknek valójában nincs szükségük a paraméterre, mégis consumer-ként viselkednek. Ezen csomópontok helyes kiválasztása az optimális kommunikáció szempontjából lényeges, de NP-teljes probléma. Ennek részleteiről a 3.2.3. fejezetben lesz szó.

A másik gond az, hogy előfordulhat olyan eset is, hogy – bár a feliratkozott csomópontok alkotta részgráf összefüggő – a friss érték soha sem jut el bizonyos csomópontokhoz, mert a szerencsétlen ütemezési sorrend miatt mindig felülíródik a régebbi értékekkel. Egy ilyen példa látszik a 16. ábrán.

Erre a problémára egy lehetséges megoldás az lenne, hogy az ütemezési sorrendet úgy kell beállítani, hogy ilyen felülírás ne fordulhasson elő. A gond az, hogy több paraméter esetén már nem biztos, hogy létezik a megfelelő ütemezés. Ehelyett a protokoll az értékkel együtt elküldi annak generációs számát is. A generációs szám ebben az esetben azt adja meg, hogy hányadszor lett az adat kiküldve. Ezt a számot aztán minden consumer is elküldi, így ha egy csomópont kap egy új paramétert, el tudja dönteni, hogy az frissebb-e, mint ami nála van, elkerülve ezzel a felülírással járó problémákat. A generációs szám lehetőséget ad arra is, hogy hogy a csomópontokon futó szoftver értesüljön a producer



16. ábra: Nem frissülő értékek. Az ábrán a piros színnel jelölt csomópont a producer, a zölddel jelöltek pedig a consumer-ek. Az időrések úgy lettek kiosztva, hogy először az A ad, aztán a B, majd a C, D és végül az E, ahogy az az ábrán is látszik. A problémát az okozza, hogy A elküldi a paraméter értékét C-nek, ezután B is elküldi a nála lévő másolatokat C-nek, ezzel felülírva a frissebb értékeket, ellehetetlenítve ezzel a legújabb értékek terjedését.

elérhetetlenségéről és megtegye a szükséges lépéseket.

A generációs számmal az a probléma, hogy előbb-utóbb túl fog csordulni. Ez másodpercenként 100 üzenetváltást feltételezve 32 bites számláló esetén 1.3 év, 16 biteset feltételezve pedig 11 perc. Mivel egy ipari vezérlőrendszernek évekig kell működnie és minden körülmények között stabilnak kell maradnia, ezért a túlcordulást le kell kezelni. Ehhez az kell, hogy a két szám összehasonlításakor ne csak a két szám viszonyát, hanem azok különbségét is vizsgálja. (Másik lehetőség lenne a számláló 64 bitesre növelése, ami még másodpercenként egymillió üzenetküldés esetén is csak 600.000 évente csordulna túl, de ez feleslegesen növelné a csomag méretét.)

Előfordulhat, hogy egy csomópont sokáig nem kap friss adatot. Ezt a generációs szám változatlanágából tudja megállapítani, de az eddigiek alapján ezzel nem törődve továbbra is multicastolja azt. Érdekes minden adathoz definiálni egy lejáratát, amit túllépve a paraméter továbbküldése leáll. Ezzel energiát lehet spórolni és a küldött csomagok méretét csökkenteni. A protokoll jelenlegi implementációja ezt nem támogatja, de a végleges termékbe érdemes ezt is beleépíteni.

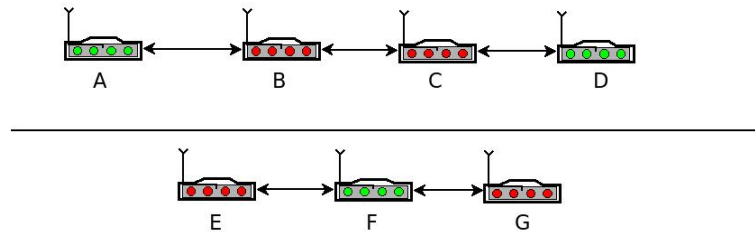
Szimultán időréshasználatot biztosító algoritmus

Ahogy arról már volt szó, a most bemutatott rendszer tipikus alkalmazási területe miatt a hálózatot reprezentáló gráf nem sűrű, ezért semmi sem indokolja, hogy egy időréshen ne adhasson több végpont egymás zavarása nélkül. Ahhoz azonban, hogy az időréseket helyesen ki lehessen osztani, szükség van egy megfelelő algoritmusra, ami az ütemezőt alkotja. Mivel a gráf felépítése csak ritkán változik (a mozgó csomópontok kiszolgálásáról később lesz szó), ezért ennek az algoritmusnak nem kell magán a hálózaton online futnia, elég egyszer, a hálózati topológia felderítésekor elvégezni.

Az ütemező prototípusa Python nyelven készült. Bemenetként a hálózati topológiának megfelelő gráfot várja, valamint azt, hogy mely csomópontok osztoznak közös paramétereken. A program első lépésben olyan minimális köröket keres, melyek lefednek minden olyan csomópontot, amik egy adott paraméterrel dolgoznak. Ez a probléma megegyezik az utazóügynök problémájával, ezért optimális megoldást nehéz rá adni, de különböző jól ismert heurisztikákkal lehet közelíteni.

A minimális körök keresése helyett üzembiztosabb megoldás a csomópontokat tartalmazó Steiner-hálózatok [30] keresése. A Steiner-hálózat probléma (vagy robosztus hálózat

tervezési probléma) során adott egy $G = (V, E)$ irányítatlan gráf, r szükséges redundancia, n maximális áteresztőképesség és u, v pontpárok. A feladat olyan minimális élszámú részgráf keresése, mely minden u, v pontpár között tartalmaz legalább r darab élfüggetlen útvonalat úgy, hogy egy adott e él nem szerepelhet útvonalként n -nél több pontpár összekötésében. Ez azt jelenti, hogy bármelyik $r - 1$ él kivétele esetén az u, v pontpárok összeköttetése megmarad és a hálózat sem terhelődik túl. A probléma NP feladat, így az optimális megoldás keresése helyett itt is csak közelítő megoldásokat lehet használni. A lineáris programozás módszere például jó közelítést ad, de ennek ismertetése túlmutat a dolgozat keretein.



17. ábra: Rejtett csomópontok problémája.[31] A B csomópont A-nak, a C csomópont pedig D-nek akar küldeni. Bár B és C szomszédok, egymás akadályoztatása nélkül képesek küldeni. E és G csomópontok egyaránt F-nek akarnak üzenni. Bár nem szomszédok, interferenciát okoznak egymásnak.

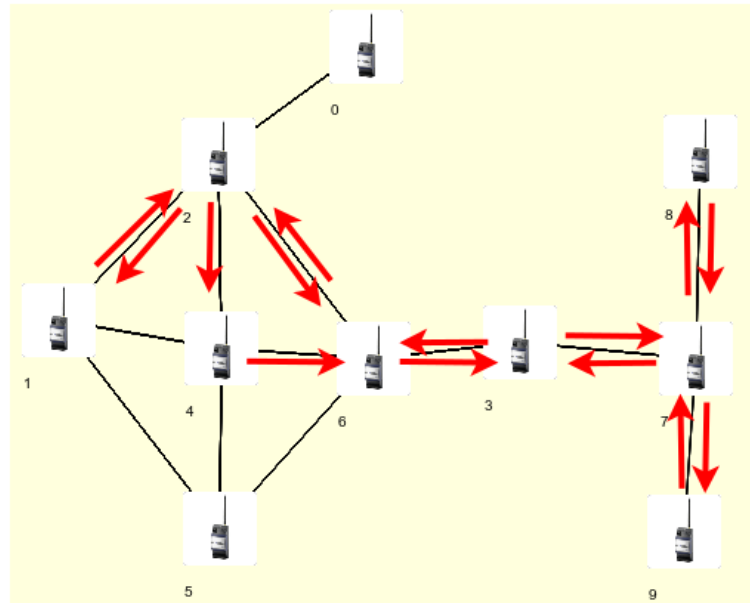
Miután a kommunikációs útvonalakat meghatározta az ütemező, ki kell osztani az időréseket. Az intuitív megoldás az lenne, hogy egy csomópont akkor kerülhet bele a T_n időrésbe, ha nincs olyan szomszédja, amelyik már benne van T_n időrésben, ez azonban interferenciát okozna a hálózaton (17. ábra).

Ennél jobb megoldás, ha egy csomópont akkor üzenhet, ha nincs olyan szomszédja, vagy második szomszédja, amelyik éppen üzen. Ezzel az interferencia biztosan megszüntethető, de a feltétel maga túl szigorú. Visszatérve a 17. ábrához, az alsó esetben a rejtett csomópont problémáját (a kép alsó része) ugyan megoldja, de a felső esetben ez sem engedélyezi az üzenet küldését.

A leghatékonyabb megoldás az, ha egy csomópont a T_n időrésben akkor adhat, ha a T_n időrésben nincs olyan szomszédja, ami egy másik csomópont címzettje. (A címzettet itt a multicast csomagok miatt úgy kell érteni, hogy nem hangzik el olyan üzenet egy másik csomóponttól, ami azt érinti.) Belátható, hogy ez a megszorítás optimális.

Miután a feltétel megvan, már csak az időréseket kell feltölteni úgy, hogy minél kevesebb legyen belőlük. Ez a jelenlegi implementációban mohó algoritmussal lett megvalósítva. Az ütemező egy időréssel kezd, amiben az első csomópont van. Ezután minden csomópont esetén megvizsgálja, hogy a soron következő belekerülhet-e valamelyik már létező időrésbe. Ha nem, új időrést kell létrehozni.

Egy ütemezési példa látható a 18. ábrán, melynek eredményeként négy időrés lett kialakítva. Miután az ütemezés elkészült, fel kell azt tölteni a hálózatra. Ehhez a globális konfigurációs üzenetekre van szükség. Érdemes még megemlíteni azt, hogy az ütemező szempontjából nem számít, hogy melyik csomópont a producer és melyek a consumer-ek, mert minden csomópontnak kell küldenie és fogadnia egyaránt annak érdekében, hogy fenntartható legyen a hálózat időszinkronizációja. ami a TDMA miatt alapkövetelmény. A fogadott üzenetek alapján minden csomópont képes szinkronizálni a saját rendszeridejét. Megfelelő időszinkronizáció hiányában az egyidőben indított, azonos hőmérsékletű és korú csomópontok (a felhasznált MSP430F2274 mikrovezérlő beépített órajelgenerátorát felhasználva) 100 másodperc alatt több másodpercet is elcsúsztak egymáshoz képest.



18. ábra: Példa egy kilenc csomópontból álló hálózatra. Két globális paraméter van, A és B. A-ra szüksége van az 1-es, 8-as és 9-es csomópontnak. B-re szüksége van az 1-es, 2-es, 6-os és 4-es csomópontnak. Az ütemező először azt generálta le, hogy melyik csomópont melyiknek küldjön. Ez látható piros nyilakkal. Végül legenerálta az időréseket. Összesen négy időrés lett: (1,3); (2,7); (8,4) és (9,6).

3.2.4. Az alkalmazásréteg

A fejezet eddigi részeiben bemutatásra kerültek a protokoll legfontosabb elemei, amiket a klasszikus OSI-modell első három rétege tartalmazott. A további rétegek funkcionalitásai a firmware implementációjához tartoznak, de az alkalmazásrétegről még érdemes itt is szót ejteni.

A TinyPLC szempontjából a kommunikációs protokoll alkalmazásrétege a végfelhasználók által írt létradiagrammal kommunikál. A protokoll célja az, hogy úgy oldja meg az egyes paraméterek átvitelét a csomópontok között, hogy a végfelhasználó csak minimális különbséget lásson a lokális és távoli paraméterek között.

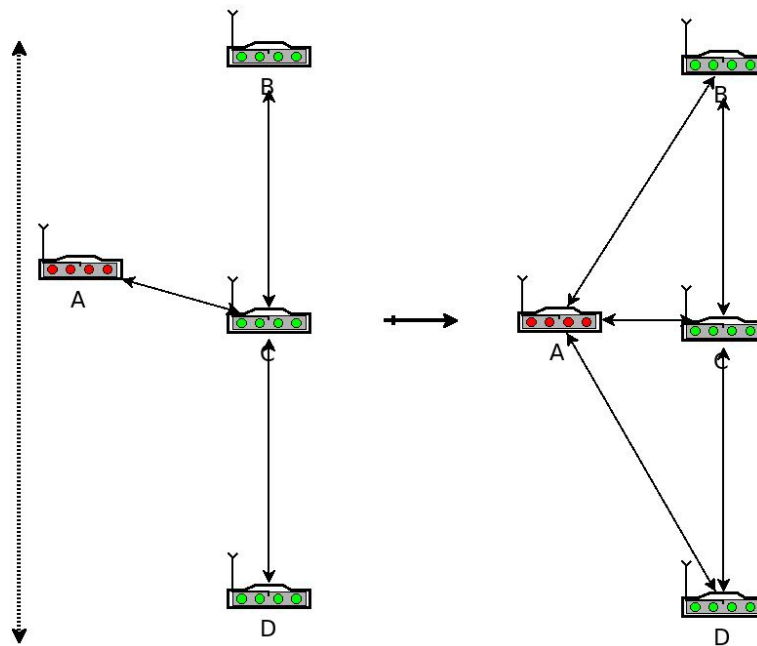
Ennek érdekében – akárcsak a fizikai ki- és bemenetek – a távoli paraméterek is le vannak képezve a memóriába. Az egyetlen különbség az, hogy az ilyen paramétereket a generációs szám követi és bizonyos alkalmazások esetén ez utóbbivan is foglalkozni kell.

3.2.5. Mozgó csomópontok kezelése

Bár a dolgozat eddigi részében csak rögzített pozíciójú csomópontokról volt szó, bizonyos megszorítások betartása mellett a protokoll alkalmas mozgó csomópontok kezelésére is.

Az alapötlet annak a kihasználása, hogy a protokoll nem küldő-fogadó párokat határoz meg, hanem állandóan ismételt multicast-üzeneteket, amiknek a forrása lényegtelen. Az egyetlen, ami számít, az a csomagban lévő adat és annak generációs száma. Így meg lehet azt tenni, hogy ahelyett, hogy a hálózat tényleges topológiáját programozzuk fel a rendszerbe, a mozgó csomópontok esetén az összes lehetséges összeköttetést fixnek vesszük, ahogy az a 19. ábrán látható.

Az A mozgó csomópont minden lehetséges szomszédjának fogyasztania kell egyrészt az



19. ábra: Mozgó csomópontok kezelése. Az ábra jobb oldalán A csomópont felle mozgást végez, egyidőben B,C és D közül csak az egyikkel tud kommunikálni. Ezt az ütemezőnek a bal oldalon látható topológiaként kell megadni.

A által fogyasztott paramétereket, hogy A mindig megkapjon mindent, másrészt A által előállított paramétereket is. Bár ezeket a tulajdonságok automatikusan is generálhatóak, az ütemező jelenlegi implementációjában erre nincs felkészítve, ezért egyelőre manuálisan kell gondoskodni róluk.

Ahogy az látható, a mozgó csomópontok helyes kezelése nagyban megnövelheti az útvonal menti csomópontok terheltségét. Éppen ezért törekedni kell arra, hogy a mozgás minél kevesebb csomópontot érintsen. Eerre az is egy lehetséges megoldás, ha nem lényeges állandó kapcsolatot fenntartani a mozgó csomópont és a hálózat többi része között, hanem bizonyos időközönként megy csak olyan régióba, ahol információt tud cserélni a hálózattal.

3.3. Fejlesztőkörnyezet

Egy ipari vezérlőrendszer piaci sikere és elterjedése elsősorban annak megbízhatóságán múlik. Az eddigiekben szó volt a csomópontok felépítéséről, a rajtuk futó firmware-ről és a kommunikációs protokollról, amik fontosak a minőségi és precíz vezérlés elkészítéséhez, de önmagukban nem elegek. Szükség van egy megfelelő, intuitív, a csapatmunkát támogató fejlesztőkörnyezetre is.

A TinyPLC rendszer használatához egy HTML alapú, szerver-kliens architektúrájú fejlesztőeszköz készült, ami összefogja a hálózati topológia megtervezését és a vezérlőszoftver megírását. A fejlesztés során a cél az volt, hogy teljesen platformfüggetlen legyen, támogassa a csapatmunkát és az egész automatizálási projektet lehessen benne kezelni.

3.3.1. A fejlesztőkörnyezet felépítése

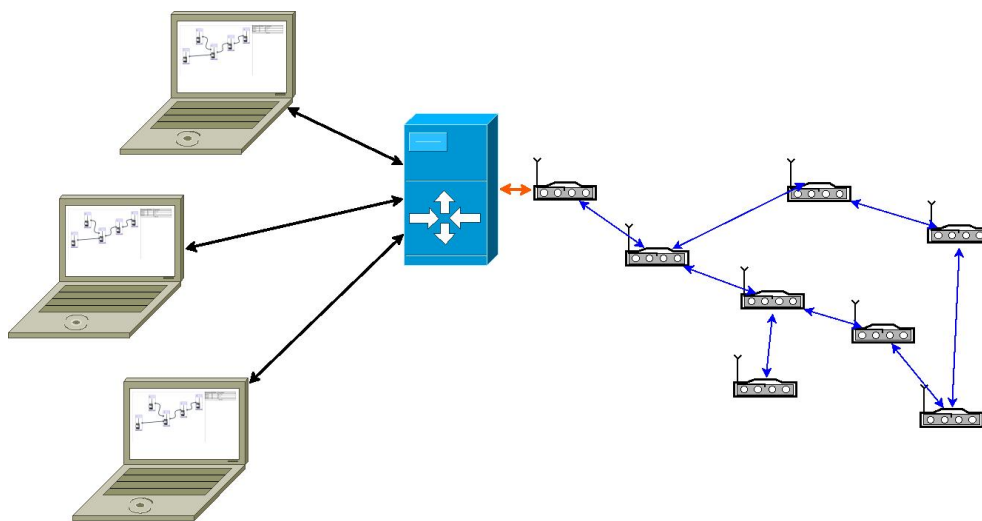
A fejlesztőkörnyezet HTML alapú, ezért először a kiszolgálót kell elindítani, ami egy standard Java alkalmazás. Ez felelős a HTTP szerver indításáért, a végpontok felprogra-

mozásáért és monitorozásáért. A Java platformra annak széles elterjedése miatt esett a választás. Az alkalmazás fejlesztése során fontos volt, hogy az elkészült szerver (amennyire ez megoldható) képes legyen futni minden asztali környezetben (Windows, Linux, Macintosh), beágyazott eszközökön (különböző onboard computerek, mint például a Raspberry Pi) vagy akár Android alapú mobiltelefonokon, tableteken is. Ez azért előnyös, mert így a rendszert bármilyen körülmények között (áramforrástól távol, nyílt terepen, kitéve az időjárási viszonyoknak) lehet fejleszteni.

A fejlesztőkörnyezet által készített programot valahogyan át kell küldeni a hálózatra. Mivel a jelenlegi prototípus AMB8423 típusú modulokra épül, ezért – a hozzá tartozó USB-soros átalakító miatt – az emulált soros vonal kézenfekvő megoldás. Az USB-port elég elterjedt, az azzal nem rendelkező eszközök pedig (mint a mobiltelefonok, vagy tabletek) általában rendelkeznek Bluetooth modullal és támogatják a SPP-t (Serial Port Profile), mellyel virtuális soros portot lehet létre hozni a telefon/tablet oldalán, amit egy olcsó IC-vel (például WLS123A1M) valódi soros vonallá alakíthat.

A Java nyelven írt kiszolgálóra a fejlesztők böngészővel csatlakozhatnak a megfelelő IP-cím és port megadásával. A csatlakozott fejlesztőket a főképernyő fogadja, ahol szerkeszthetik a hálózat topológiáját, láthatják a globális paramétereket és meg tudják nyitni az egyes modulok vezérlőkódját. Mivel a környezet csak szabványos HTML-elemekből épül fel, ezért azt akár egy tableten keresztül is meg lehet tekinteni, vagy éppen szerkeszteni.

A fejlesztőkörnyezet sematikus felépítése a 20. ábrán látszik.



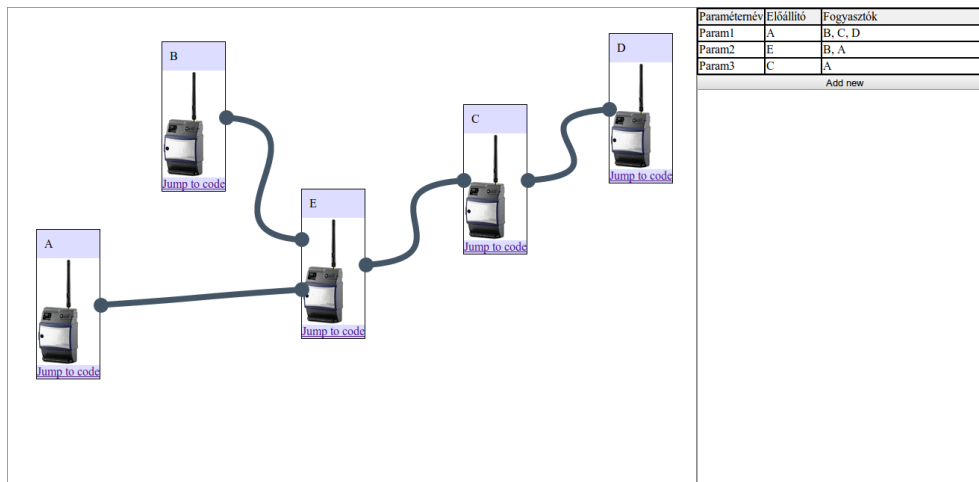
20. ábra: A fejlesztőkörnyezet vázlatos felépítése. Feketével az Ethernet/IP hálózat, narancssárgával a soros kommunikáció, késsel pedig a 3.2. fejezetben bemutatott protokoll látható.

3.3.2. A felhasználói interfész

A felhasználói interfészek helyes és intuitív megtervezése külön irodalommal rendelkezik. A jelenlegi dolgozat nem is törekszik arra, hogy egy olyan rendszert mutasson be, ami minden ergonómiai igénynek megfelel, a cél inkább egy vázlat bemutatása, ami a szükséges funkciókat tartalmazza és ami alapján létre lehet hozni egy teljes értékű környezetet.

3.3.3. A kezdőképernyő

A felhasználó a megfelelő cím begépelése után a kezdőképernyőn találja magát (21. ábra), ahol a projektről kaphat egy áttekintő képet.



21. ábra: A fejlesztőkörnyezet nyitóképernyője. Bal oldalon a projectben szereplő végpontok láthatóak, jobb oldalt pedig a globális paraméterek.

A képernyő két részre osztható. Bal oldalt a projektben szereplő végpontok és összeköttetések szerepelnek. Itt lehet új modulokat hozzáadni, régieket törölni és a közöttük lévő összeköttetéseket megadni. A modulok képe alatti linkekre kattintva az adott modul vezérlőkódja jelenik meg létradiagram képében. Jobb oldalt a globális paraméterek és azok adatai láthatóak. A táblázat alján lévő gombbal lehet új paramétereket felvenni.

Ez a képernyő JQuery-t és jsPlumb-ot használ. Mindkettő szabadon felhasználható JavaScript könyvtár, előbbi egy jól ismert és általánosan használt könyvtár, utóbbi pedig gráfok bevitelét, kezelését és kirajzolását teszi lehetővé. Az oldal feladatai közé tartozik az is, hogy AJAX segítségével tartsa a kapcsolatot a szerverrel, elküldje vagy kérje a modulok forráskódját.

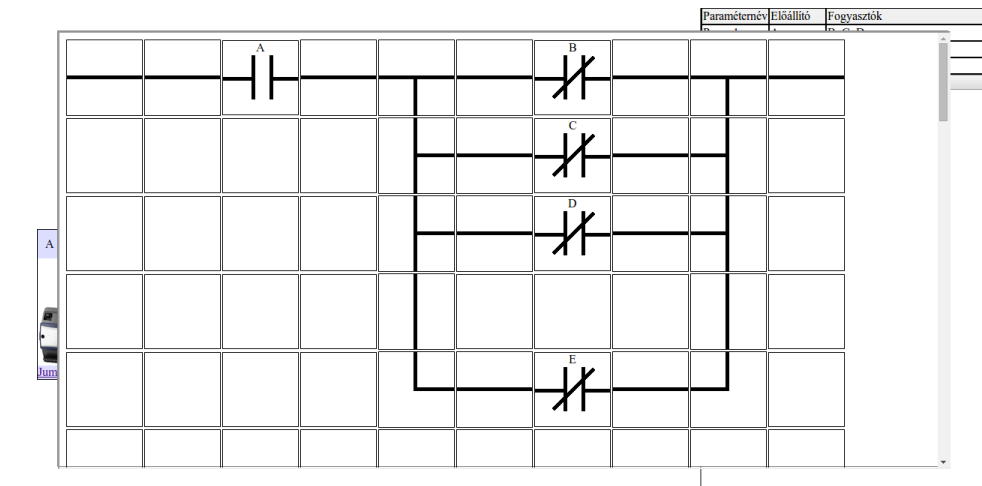
3.3.4. Kódszerkesztő

A kezdőképernyőn a modulok képe alatti linkekre kattintva meg lehet nyitni a kódszerkesztőt, melyről egy példa látható a 22. ábrán.

A kódszerkesztőben a fejlesztőnek lehetősége van használni az alapvető vezérlési struktúrákat és építőelemeket. A képernyőről az Escape gomb megnyomásával lehet menteni, majd visszajutni a kezdőképernyőre. Ez a képernyő nem csak megjeleníti és szerkeszti a létradiagramot, hanem le is fordítja a 3.1.3. fejezetben bemutatott formátumra, így a kiszolgálón (ami nem feltétlenül erős) már nem kell elvégezni a fordítást.

3.3.5. A fordító algoritmus

A fordító algoritmusának alapötlete, hogy az egyes rung-ok elejéről indulva utasításról utasításra végighalad és meghatározza annak szöveges reprezentációját, amíg elágazáshoz nem ér. Elágazás esetén megkeresi az összes részrungek kezdőelemét, rekurzívan meghívja magát rájuk, az eredményeket összefűzi az elágazásnak megfelelő formátumra, majd az elágazás után folytatódik az algoritmus.



22. ábra: A kódszerkesztő.

Az egyes utasításokat osztályok reprezentálják, amik kezelik annak grafikus megjelenítését, beállítási lehetőségeit, valamint az utasítás lefordítását is. Az utasításokat reprezentáló osztályok prototípusa az 1. forráskódban látható.

```

1 function BaseElement(){
2   this.coord=[0,0];
3   this.draw=function(){};
4   this.setup=function(){};
5   this.toCode=function(){return ""};
6   this.getName="";
7 }

```

1. Listing: Az utasításokat reprezentáló osztályok prototípusa

A tényleges fordítóalgoritmus nem tárol az utasításokról semmilyen információt, mindent a hozzá tartozó osztályokból nyer ki. Ennek eredményeként a fordítást végző függvény egyszerű (összesen 37 sor) és új utasítás bevitele esetén nem szükséges módosítani.

3.3.6. A kiszolgáló

A kiszolgáló egy Java-alapú konzolos program, mely tárolja a projekt forráskódját, biztosítja az osztott hozzáférést a projekthez (egy végpont forráskódját egyszerre csak egy fejlesztő szerkesztheti) és gondoskodik arról, hogy az elkészített konfiguráció és program rákerüljön a hálózatra. (Ehhez globális és lokális konfigurációs üzeneteket használ.)

Fontos, hogy valamilyen formában (jelenleg soros porton) össze legyen kötve egy megfelelően felprogramozott végponttal, amin keresztül a hálózatba ki tudja küldeni a lokális és globális konfigurációs üzeneteket. (Mivel a rendszer protokollja TDMA-alapú, ezért a kiszolgáló nem adhat ki bármikor csomagokat, csak a számára fenntartott időrésben. Ehhez precíz időzítésre van szükség, amire a hagyományos operációs rendszerek képtelenek, ezért kell a megfelelően előkészített végpont.)

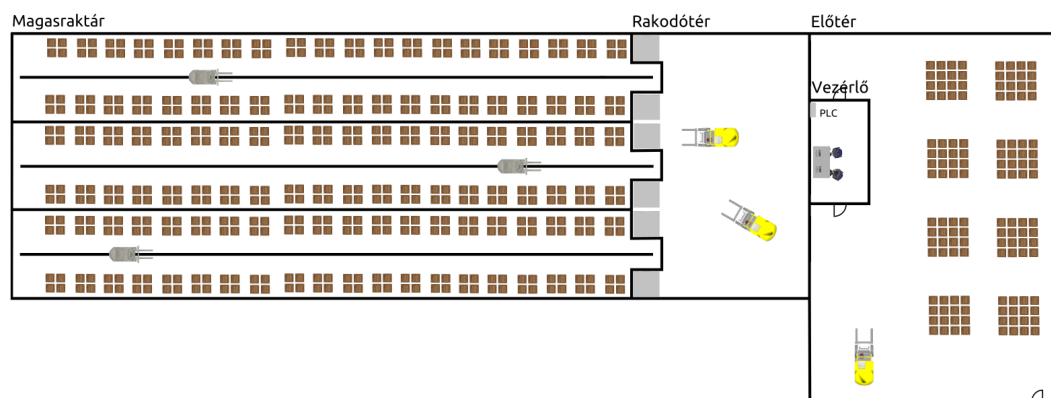
4. Esettanulmány - Automatizált felrakógép

Minden gyártási folyamat során fontos, hogy az alapanyagokat és a készterméket tárolni lehessen, szükség esetén pedig gyorsan hozzá lehessen jutni. Az ezt kezelő logisztikai rendszerek komplexitása a gyár méretétől és jellegétől függően széles skálán mozoghatnak az tisztán emberi munkaerőtől a teljesen automatizált AGV (automated guided vehicle) rendszerekig.

Ebben a fejezetben egy Magyarországon működő, gyógyszeripari alapanyagokat raktározó automatizált felrakógép kerül bemutatásra, ahol a vezérlés egy Rockwell PLC és három Ethernet/IP-n keresztül vezérelhető IO egység segítségével lett megvalósítva. Bár a helyszínen nem történt meg a dolgozatban bemutatott rendszer kipróbálása, a telepítéshez szükséges mérések el lettek végezve, ezért azt össze lehet hasonlítani a mostani megoldással.

4.1. A helyszín bemutatása

A most bemutatásra kerülő kötőtpályás felrakógép (mely a 23. ábrán látható) egy 50 méter hosszú, 16 méter széles és 20 méter magas, teljesen automatizált raktárhelyiségben üzemel, ahová a (nem automata) targoncák három beadóponton keresztül vihetnek be dobozokat és három felvevőpontra keresztül kaphatják vissza azokat. A logisztikai rendszer adatbázisában minden dobozt egy vonalkód azonosít, amihez el van tárolva a raktáron belüli pozíciója is. Ez utóbbit négy számmal (felrakógép sorszáma, oldal, sor, oszlop) lehet megadni. Egy új doboz beadása esetén a targonca vezetője egy kézi leolvasóval lehúzza a rajta lévő vonalkódot, elviszi a szállítmányt egy beadóponthoz és az adott ponthoz tartozó felrakógép felveszi és eljuttatja a raktározási helyére. A raktározott termékek kivétele ehhez hasonlóan történik, a vezető lehúzza a kézi leolvasójával a nála lévő vonalkódtömb következő elemét, a felrakógép kihozza a kért dobozt a felvevőpontra és a targonca már viheti is azt a végleges helyére.

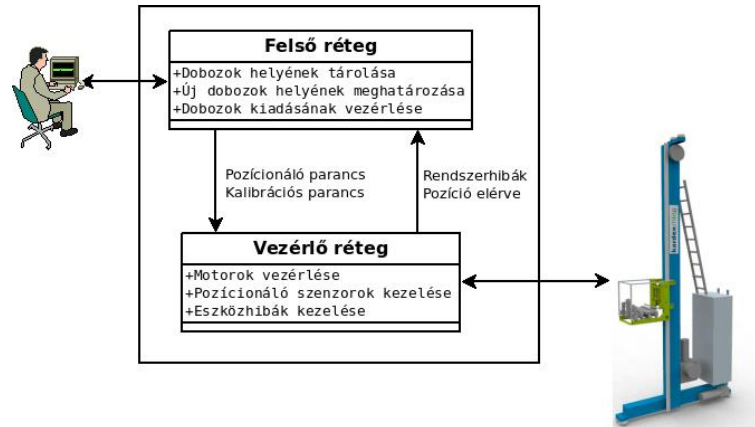


23. ábra: A kötőtpályás, automatizált felrakógép alaprajza. Jobb oldalt az előtér látható, benne a berakodásra váró dobozokkal és a vezérlőteremmel. Bal oldalon a raktár, ahol az automatikus felrakógépek működnek.

A rendszer késztermékeket és alapanyagokat egyaránt tárol, ezért a gyártás szempontjából kritikus jelentősége van: meghibásodása esetén teljes gyártási folyamatok állhatnak meg, ami komoly bevételkiesést okoz, éppen ezért fontos, hogy a rendszer minél ellenállóbb legyen minden fajta hibával szemben.

4.2. A működés leírása

A rendszer vezérlése (ahogy az a 24. ábrán is látszik) kétszintű. A felső szint felelős a dobozok adatainak tárolásáért, az újonnan bevitt egységek helyének meghatározásáért, a vonalkódleolvasók kezeléséért és az adatbázis karbantartásáért. Ez a réteg (amit számítógépeken futó programok összessége alkot) tartja az operátorral a kapcsolatot és adja ki a vezérlőrétegnek az utasításokat.



24. ábra: A felrakógép vezérlésének sematikus ábrája

A vezérlőréteg (ami a PLC-ben van megvalósítva) az, ami a felrakógépek mozgásáért felelős. Itt történik a szenzorok feldolgozása és a motorok vezérlése. Van néhány olyan biztonsági szenzor, ami nem a vezérlőrétegen keresztül, hanem közvetlenül áramkörileg fejt ki hatását (például a felrakógép azonnali leállítását). Ilyenek a vezetőfülkében és a pálya végén található vészgombok, a motorvédő kapcsolók, valamint a pálya végét jelző mechanikus kapcsolók, amik a sínről történő lefutást akadályozzák meg.

Az operátor három üzemmódban tudja vezérelni a felrakógépeket:

- i.) **Automatikus üzemmódban** a rendszer nem igényel emberi beavatkozást, csak felügyeletet. A felső vezérlési szint a vonalkódok alapján a megfelelő pozícióba küldi a felrakógépeket és kezeli a ládák helyzetét.
- ii.) **Félautomata üzemmódban** az operátornak az operátorszobából a számítógépén keresztül kell megadnia, hogy a felrakógépek pontosan hová menjenek. Akkor kell erre az üzemmódra váltani, ha a felső réteg valamiért meghibásodott (érvénytelen adatbázisállapot, hálózati hiba, stb.), de a vezérlőréteg teljesen működőképes.
- iii.) **Manuális üzemmódban** az operátor a felrakógépek pilótafülkéjéből vezérli azokat a manuális karok és gombok segítségével. Ez az üzemmód akkor használható, ha kritikus szenzorok hibásodtak meg, de a motorok vezérlése működőképes. Ez az üzemmód nem kerüli meg a vezérlőréteget, csak kikapcsolja annak bizonyos funkcióit. (A sínpálya vége felé bevezetett sebességkorlátozás például így sem kerülhető meg.)

Az üzemmódok alapján látszik, hogy a vezérlőréteg a gyártás szempontjából kritikus. Amennyiben ez a rendszer meghibásodik, a felrakógépeket nem lehet tovább üzemeltetni. A továbbiakban csak erről a rétegről lesz szó.

4.3. A vezérlőréteg

A vezérlőréteget két aspektusból lehet vizsgálni. Egyrészt a program felépítése alapján, másrészt annak fizikai felépítése szerint. Mindkét szempontból össze lehet hasonlítani a dolgozatban bemutatott rendszerrel.

4.3.1. A rendszer fizikai felépítése

A 23. ábrán látható a rendszer alaprajza, amin látszik az operátori szoba, benne (a bal felső sarokban) a PLC-t tartalmazó kapcsolószekrénnel. A szekrényben lévő PLC Ethernet/IP-n keresztül csatlakozik a felrakógépekben lévő IO-egységekhez, amik a gyár többi részétől egy fizikailag is elkülönülő hálózaton vannak.

Amennyiben az IO egységek és a PLC közötti hálózat megsérül, az érintett felrakógépek működésképtelenné válnak, ezért a kapcsolat megtervezésére nagy figyelmet fordítottak, több módszert is kipróbáltak a fejlesztés során. A dolgozat szempontjából a legérdekesebb ilyen kísérlet egy WiFi-s hálózat beüzemelése volt, amit el kellett vetni, mert túl sok volt a hálózati hiba, részben a nagy távolságok, részben a frekvenciaváltók és motorok miatti elektromos zaj miatt. Végül a felrakógépek moduljai a mennyezeten keresztül vezetett, mozgó kábelsínekben keresztül lettek hozzákapcsolva az Ethernet-hálózathoz. A módszer hátránya, hogy érzékeny a kábel meghibásodására, amit redundanciával lehet ugyan javítani, de a meghibásodott kábelek cseréje nagyon munkaigényes folyamat.

Egy karbantartás során a választott központi felépítés helyett felmerült egy szigetüzemű megvalósítás is, amelyben minden felrakógépre lett volna telepítve egy PLC, amik közvetlenül kapták volna az utasításokat a felsőbb rétegtől. Ez a megvalósítás sokkal robosztusabb lett volna, hiszen a felrakógépek manuálisan akkor is tudtak volna működni, ha minden más részegység meghibásodik, ezen kívül sokkal precízebb vezérlést lehetett volna megvalósítani, nagyobb mozgási sebességet érve el ezzel. A szigetüzemű felépítés a túlzottan megnövekedett költségek miatt lett elvetve.

4.3.2. A program felépítése

A PLC-ben futó szoftvernek három, alapvetően ugyan olyan felépítésű, egymástól teljesen független felrakógépet kell vezérelnie, melyeket három különálló, egymástól csak kismértékben különböző program valósít meg.

A programok egyenként 132 darab logikai, 7 darab 32 bites egész típusú, 34 darab 16 bites egész típusú, 14 darab 8 bites egész típusú és 54 időzítő típusú változót tartalmaz, ami összesen 289 bájtnyi adatot jelent.

A három gép vezérlésén kívül van egy negyedik program is, ami összefogja a felrakógépek vezérlését. Itt történik az utasítások szétosztása és a kommunikációs hibák lekezelése. Szerkezete sokkal egyszerűbb, mint a másik három programé.

4.4. Vezérlés megvalósítása TinyPLC rendszerrel

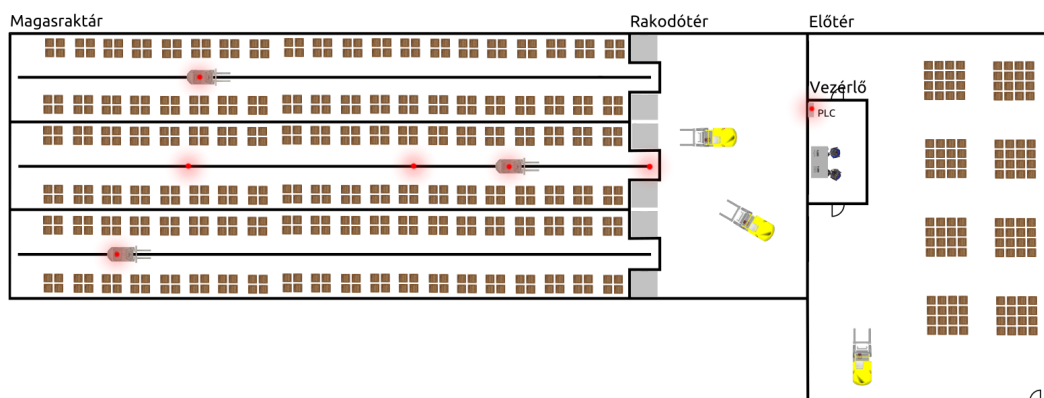
A jelenlegi megvalósítás fő problémája, hogy a központi vezérlés vagy egy hálózati hiba miatt működésképtelenné válhat az összes felrakógép. (Hálózati hiba nem csak kábelszakadás miatt történhet, hanem akár egy rosszul beállított eszköz miatt is.) A másik (bár lényegesen kisebb) gond az, hogy a PLC és a beavatkozó szervek közötti hálózati

késleltetés korlátozza a megvalósítható vezérlés pontosságát. Érdeemes lenne központiról átalakítani sziget üzeműre, de a plusz PLC-k és a hozzájuk szükséges eszközök beszerzése nagy költségekkel jár. A helyszínen végzett mérések alapján a dolgozatban bemutatott rendszer ideális lenne a felrakógép vezérlésének a megvalósítására.

4.4.1. A végpontok elhelyezése

Az első feladat a végpontok helyének megfelelő kiválasztása. Ennek érdekében a helyszínen különböző méréseket végeztem a vezeték nélküli kommunikációs lehetőségek felderítése érdekében. A feladat egy olyan csatorna kiépítése volt, ami az operátorszobában lévő modult (ez a modul tartja a kapcsolatot a felső réteggel) összeköti a felrakógépeken lévő modulokkal (amik a tényleges vezérlést végzik).

A javasolt elrendezés a 25. ábrán látható. Az operátorszobából megbízhatóan kijut a jel a középső sínek fölé, ahol egy végpont továbbküldi azt a sín feletti maradék kettőnek. A középső és a két szélső felrakógép mindig elég közel lesz legalább az egyik jelerősítőként funkcionáló végponthoz ahhoz, hogy megbízhatóan tudjanak kommunikálni egymással.



25. ábra: A csomópontok javasolt elhelyezése: Az operátorszobában egy, a középső sín felett három és minden felrakóg tetején egy.

4.4.2. A vezérlőszoftver portolása

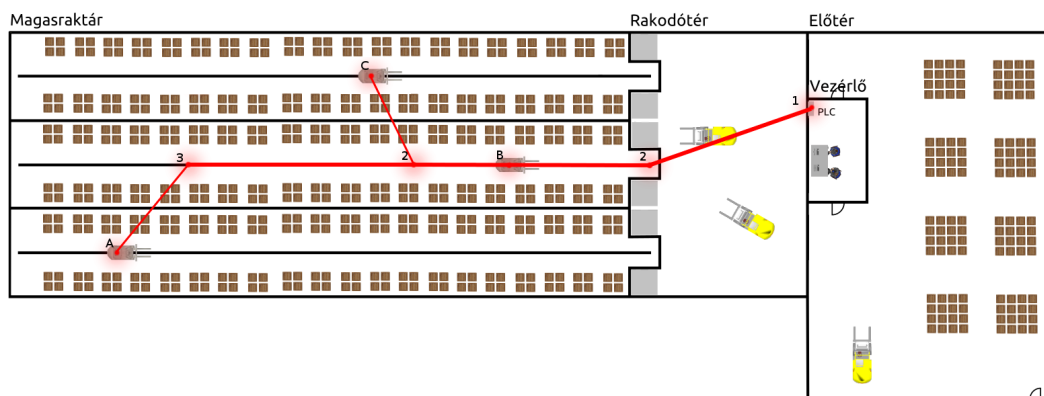
A felrakógépek vezérlése külön-külön nem bonyolult feladat, a szükséges adatok tárolásához még a memóriakártyát sem kell használni, elegendők hozzá a prototípus mikrovezérlőjén található erőforrások.

A program, mivel nagyrészt létradiagramban van írva, szintén könnyen portolható a most bemutatott rendszerre. A felső réteggel kapcsolatot tartó részt az operátorszobában lévő modulnak kell futtatnia, ami elküldi a felrakógépeknek a kívánt pozíciót. A felrakógépeken lévő csomópontok a kívánt pozíció értékét kapják meg és az aktuális pozíciót, valamint egy, a hibákat tartalmazó státuszt küldenek vissza az operátorszobába.

4.4.3. A kommunikációs topológia

A helyszínen elvégzett mérések alapján a 26. ábrán látható kommunikációs útvonalak mentén lehet összekötni a rendszert.

Az ábrán látható csomópontoknak három szerepük van:



26. ábra: A csomópontok között kialakult kommunikációs útvonal.

- az 1-es számmal jelölt csomópont a felsőbb réteggel és az operátorral tartja a kapcsolatot, a felrakógépeknek kiküldi az elérendő pozíciót;
- a 2-es, 3-as és 4-es számmal jelölt csomópontok csak átjátszóállomásként működnek;
- az A, B és C betűvel jelölt csomópontok a felrakógépeket vezérlik, a hozzájuk legközelebbi átjátszóállomásokon keresztül tartják a kapcsolatot az 1-es csomóponttal.

Az ütemezőt futtatva a hálózatra az alábbi időrések jöttek létre: {1,4}; {2}; {3}; {A}; {B}; {C}. Ebben a konkrét példában az időrések megosztásával nem lehetett sokat spórolni, mert a mozgó csomópontok miatt a hálózat magasraktárban lévő része erősen összefüggő.

Az eredményül kapott vezeték nélküli vezérlőhálózat az 1-es csomópont és a legtávolabbi (4-es eszközön keresztül kommunikáló) felrakógép közötti késleltetés az iránytól és az ütemezési sorrendtől függ. Egy lehetséges ütemezés például a már megadott {1,4}; {2}; {3}; {A}; {B}; {C} sorrend. Ekkor, ha C felrakógép a 4-es node-on keresztül kommunikál, az 1-es és C közötti késleltetés 6 időrésnyi lesz, az ellenkező irányba 8. Mivel egy időrés jellemzően ezredmásodpercekben mérhető, ezért ez a késleltetés az alkalmazás szempontjából elhanyagolható.

Az esettanulmány alapján érdemes lenne a jelenleg meglévő központi vezérlést lecserélni a TinyPLC rendszerre a felrakógépek esetén, hiszen így:

- a felrakógépek legalább kézi üzemmódban mindig működnének, növelve ezzel a jelenlegi rendszer megbízhatóságát;
- a helyi vezérlés gyorsabban reagálhatna a felrakógép visszajelzéseire, ezzel nagyobb sebességet tenne lehetővé;
- az Ethernet-kábeleket nem kellene átvinni a mozgó felrakógépekre.

5. Szimuláció és validáció

A korábbi fejezetekben bemutatott protokoll tesztelése történhet valós mérések elvégzésével vagy valamilyen szimulációs eszköz segítségével. A szimulátoroknak több előnye is van:

- **egyszerűen és gyorsan** lehet lecserélni vagy módosítani a végpontokon futó programot, nem kell minden módosítás után a kihelyezett csomópontokat begyűjteni, manuálisan újraprogramozni és újra kihelyezni;
- a szimulátor sokkal **költséghatékonyabb**, akár több ezer csomópont szimulálása is megoldható anyagi befektetés nélkül;
- a **mérések könnyen elvégezhetőek**, a szimulációs környezetek bármilyen paramétert képesek tárolni.

Azt is figyelembe kell venni, hogy egy szimuláció lehet téves is, ezért a szimulációs eredményeket validálni kell. Jelen fejezetben először a szimulációs környezet és a kapott eredmények kerülnek bemutatásra, majd a kapott eredmények valós mérésekkel lesznek összehasonlítva.

5.1. A szimuláció célja

A szimuláció során számtalan tulajdonságot lehet vizsgálni, melyek látjogosultságát és fontosságát az alkalmazás határozza meg. A most bemutatásra került rendszer két kritikus paramétere a *késleltetés* és az *áteresztőképesség*. A szimulálás kezdete előtt meg kell határozni az ezekhez szükséges mérőszámokat.

A késleltetés alacsonyan tartása alapvető fontosságú egy valós idejű vezérlőrendszer rendszer kommunikációja során. Jelen esetben az alábbi három komponensből áll össze:

- a végpontok közötti **rádiós kapcsolat sebessége** korlátozott;
- a továbbításban részt vevő végpontok nem képesek azonnal továbbítani a fogadott csomagokat, a **feldolgozási sebességük** véges;
- mivel **a protokoll időosztás-alapú**, a végpontok a fogadott csomag feldolgozása után nem üzenhetnek egyből, meg kell várniuk, míg lesz szabad időrésük.

Mivel a csomag átvitelének és feldolgozásának bele kell férnie egy időrésbe, ezért két csomópont közötti késleltetés meghatározásához azt kell megmérni, hogy a feladás és a fogadás között hány időrés telt el.

Az áteresztőképesség mérése során azt kell vizsgálni, hogy a végpontok mennyi csomagot képesek küldeni és fogadni egy adott időtartam alatt. Ez akkor fontos, ha például valamilyen mért értéket nagy mintavételezéssel kell naplózni. (Ilyenkor kevésbé számít az, hogy az adatok mennyi idő alatt jutnak el a mérési ponttól a naplózás helyéig.) Az áteresztőképesség meghatározásához azt kell mérni, hogy a címzett milyen gyakran kap csomagokat, ha a feladó folyamatosan ad.

A késleltetés és az áteresztőképesség vizsgálatát nem elég egyetlen elrendezésben elvégezni, ellenőrizni kell, hogyan változnak a topológia és a végpontok számának függvényében. Mivel a protokoll szerves része az ütemező, azt is érdemes mérni, hogy a használata mennyire képes javítani a két vizsgált paramétert.

5.2. Szimulátorok bemutatása

A szimuláció megkezdése előtt ki kell választani a tesztkörnyezetet, amivel a legegyszerűbben lehet elvégezni a szükséges méréseket.

- A **Matlab** egy magasszintű programozási eszköztár numerikus számítások elvégzésére és az eredmények vizualizálására. Bár igen sokrétű és implementálva is van benne több WSN protokoll is [32], sokkal általánosabb célú eszköz annál, mint amire a dolgozatban bemutatott protokoll során szükség van.
- Az **Omnet++** egy bővíthető, moduláris, komponens alapú C++ szimulációs eszköztár, ami hálózatok szimulálására lett létrehozva. Előszeretettel alkalmazzák kommunikációs, áramköri, logisztikai vagy akár chipen belüli hálózatok modellezésére. Rengeteg bővítmény található hozzá. Mivel nem konkrétan WSN szimulálására lett kifejlesztve, ezért a megfelelő környezet beállítása nem egyszerű. Az OMNeT++ használata helyett WSN szimuláció esetén érdemesebb inkább a Castalia-t használni, ami egy erre a célra kifejlesztett modul.
- A **Castalia** egy modul az OMNeT++-hoz, célja egy hatékony eszközt nyújtani a WSN szimulálásához. Támogatja a különböző csatornamodelleket, a mozgó csomópontokat, a csomópontok közötti változó jelerősséget, interferenciát és még sok egyéb jelenséget, amit egy WSN fejlesztése során figyelembe kell venni. Nagyon jól lehet vele mérni a WSN különböző jellegezetességeit. Használata során probléma volt a C nyelv használata. Típusossága megnehezítette a csomagok és egyéb adatszerkezetek gyors átalakítását és lassította a szimulációt.

5.3. Saját szimulátor bemutatása

Mivel az elérhető szimulátorok között nem találtam olyant, ami mindenben megfelel az igényeimnek, a rendszer hatékony teszteléséhez készítettem egyet. A tervezés során az alábbi szempontok domináltak:

- statikus hálózatok egyszerű és gyors modellezése;
- tetszőleges topológia intuitív bevitele;
- dinamikus adatszerkezeteket támogató scriptnyelv biztosítása;
- valós csomópontokhoz hasonlóan legyen:
 - periodikus megszakítás, ami adott időközönként fut le;
 - üzenetfogadó megszakítás, ami minden üzenet fogadásakor lefut.
- összeköttetések minősége állítható legyen;
- valós, de egyszerű csatornamodell;
- alkalmas legyen az algoritmus szemléltetésére;
- alkalmas legyen különböző mérések elvégzésére;
- a használata minimális előképzettséget és előkészületet igényeljen.

Ennek eredményeként egy HTML-alapú szimulátor jött létre, melyet JavaScript nyelven lehet programozni.

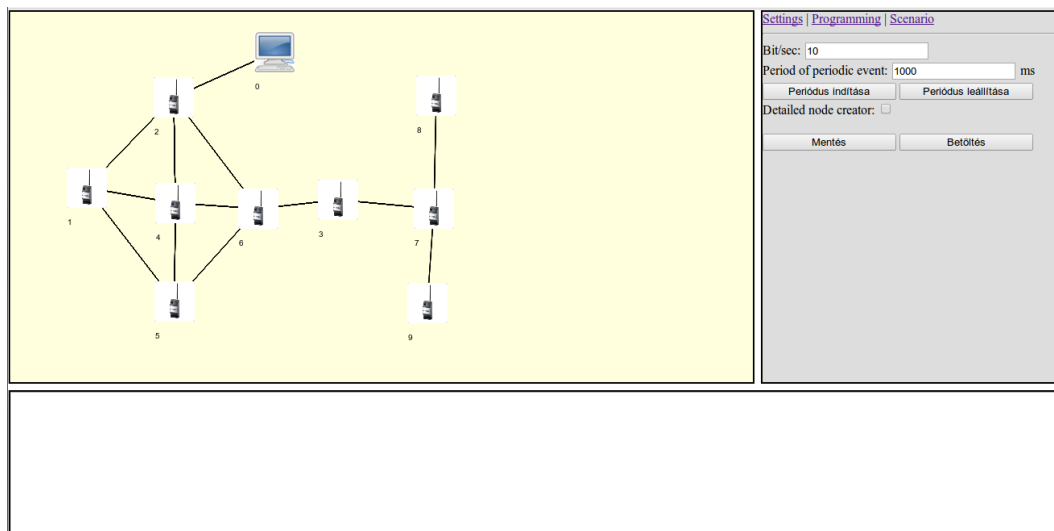
5.3.1. A szimulátor kezelőfelülete

A szimulátor kezelőfelülete a 27. ábrán látható. Középen a WSN csomópontjai és összeköttetései láthatóak. Új csomópont felvételéhez rá kell kattintani a csomópont helyére és végig kell menni a felugró lépéseken. A csomópontok gyors létrehozása érdekében lehetőség van arra, hogy létrehozáskor a rendszer csak a csomópont nevét kérje el, minden mást hagyjon alapértelmezettként. (Ezt a funkciót a jobb oldali kezelőpulton lehet aktiválni.) Alul a konzol látható, amin keresztül a szimuláció során a program különböző információkat jeleníthet meg. A konzolra történő kiíratás nincs korlátozva.

A bal oldalon a kezelőpult látható, aminek három füle van: Settings, Programming és Scenario. (28. ábra) A *Settings* fülön belül a szimulációra és kezelőfelületre vonatkozó beállításokat lehet szabályozni. Lehet állítani a bitsebességet, a periodikus megszakítás sebességét, el lehet indítani és lehet szüneteltetni a periodikus eseményt, valamint el lehet menteni és vissza lehet tölteni a projektet.

A *Programming* fülön lehet implementálni a JavaScript nyelven az üzenetkezelő és a periodikus megszakítást. Mindegyik szövegmezőben lehet függvényeket és változókat létrehozni, így könnyen és gyorsan megvalósítható tetszőleges funkció.

Végül a *Scenario* fülön a csomópontokat és az összeköttetéseket lehet definiálni. Új csomópont létrehozása esetén ide automatikusan bekerül az azt létrehozó kód is.



27. ábra: A szimulátor kezelőfelülete. Alul látható a konzol, amire futás közben lehet kiíratni bármit, középen az eseménytér, ahol a WSN csomópontjai láthatóak, jobb oldalon pedig a kezelőpanel.

```

Settings | Programming | Scenario

Üzenetkezelő kód (üzenet tartalma: msg)
function configure(msg){
  if (msg.subtype=="SLOTCONF"){
    writeToConsole(self.id+": SLOTCONF
fogva");
    self.tmpSlotNum=msg.slotNum;
    self.tmpMySlot=msg.slots[self.id];
    self.freeze=msg.freeze;
  }
}

if (self.id==0){

Periodikus kód (periódus száma: count)
if (self.id==0){
  return; // Felügyeleti. Ezen keresztül
  avatkozik be a user
}

if (count==1){
  // Inicializáció
  self.slotNum=1; // Kezdetben egy slot
  van, a konfigurációs üzeneteknek
  self.mySlot=1; // Más üzenet nincs
  self.configMsgQueue=[];
}
}
Programming

```

```

Settings | Programming | Scenario

addNode('1', 70, 193, 50, 50, undefined);
addNode('2', 178, 81, 50, 50, undefined);
addNode('4', 180, 212, 50, 50, undefined);
addNode('5', 179, 334, 50, 50, undefined);
addNode('6', 283, 219, 50, 50, undefined);
addNode('3', 382, 208, 50, 50, undefined);
addNode('7', 501, 219, 50, 50, undefined);
addNode('8', 504, 79, 50, 50, undefined);
addNode('9', 493, 336, 50, 50, undefined);
connect(1,2);
connect(2,4);
connect(5,4);
connect(5,1);
connect(4,6);
connect(2,6);
connect(5,6);
connect(3,6);
connect(3,7);
connect(8,7);
connect(9,7);
connect(4,1);
addNode('0', 303, 25, 50, 50,
"computer.png");
connect(0,2);

```

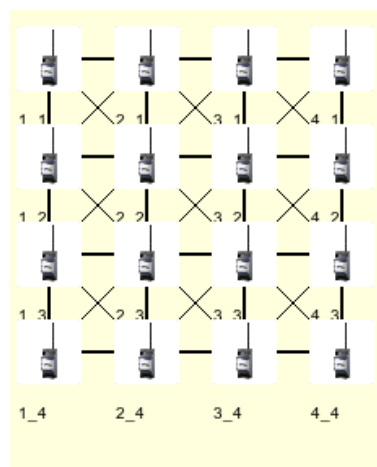
28. ábra: A fenti ábrán a kezelőpult maradék két ablaka látható: **Programming** és **Scenario**.

5.4. Protokoll tesztelése a szimulátorral

A szimuláció során a rendszer teljesítőképesség-analízise négyzetrácsos és soros elrendezésben történt meg. Mindkét esetben megtörtént a késleltetés és az áteresztőképesség vizsgálata.

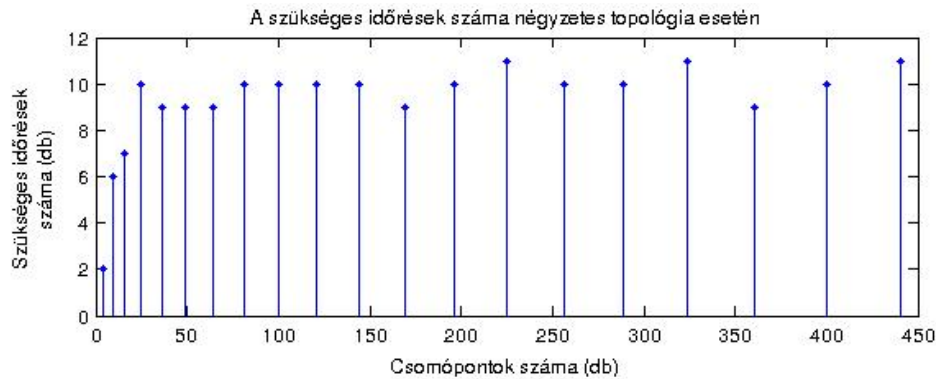
5.4.1. Négyzetrácsos elrendezés

A négyzetrácsos elrendezés során a modulok egy $n \times n$ méretű négyzetrács csomópontjaiba kerültek elhelyezésre. A modulok tudnak kommunikálni a vízszintes, függőleges és átlós szomszédaiikkal. Egy példaelrendezés látható a 29. ábrán. Ez az elrendezés szimulációk során gyakori, mert jól közelíti az egyenletesen szétszórt csomópontokat. A szimuláció során az összes csomópont generált olyan adatot, melyre a hálózat többi tagjának szüksége volt. A késleltetés a bal felső és a jobb alsó csomópont között lett mérve. Mivel a késleltetés már egyszerűbb esetekben is (lásd 4.4.3. alfejezet) függhet az időrések sorrendjétől, egy adott ütemezéssel több mérés is készült.



29. ábra: **Négyszer négyes elrendezés.** A négyzetrácsos elrendezés során az egymás melletti csomópontok vízszintesen, függőlegesen és átlósan tudnak kommunikálni.

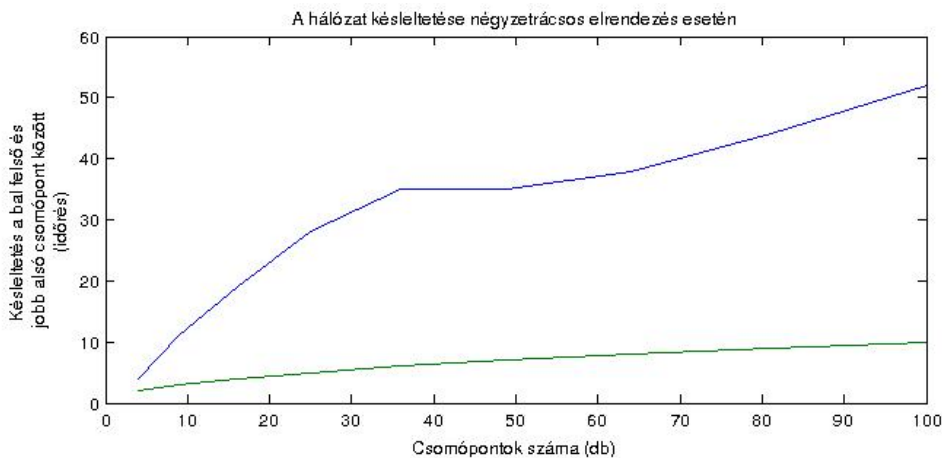
Első lépésben az áteresztőképesség meghatározása történt meg, amihez az időrések számát kellett meghatározni. A kapott eredményeket szemléltető grafikon a 30. ábrán látható.



30. ábra: Az időrések száma a csomópontok számának függvényében, négyzet-rácsos elrendezés esetén.

A grafikon viszonylag gyorsan ($n = 5$, vagyis 25 csomópont esetén) eléri a 10 időrést, ezután 12 és 9 között váltakozik. Ez azt jelenti, hogy a mérést végző csomópontok tetszőleges hálózati méret esetén átlagosan 10 időréseként képesek adni. Az áteresztőképesség egy bizonyos határ felett (25 végpont) már nem függ a hálózat méretétől. Ez a szimuláció az ütemező létjogosultságát is igazolja.

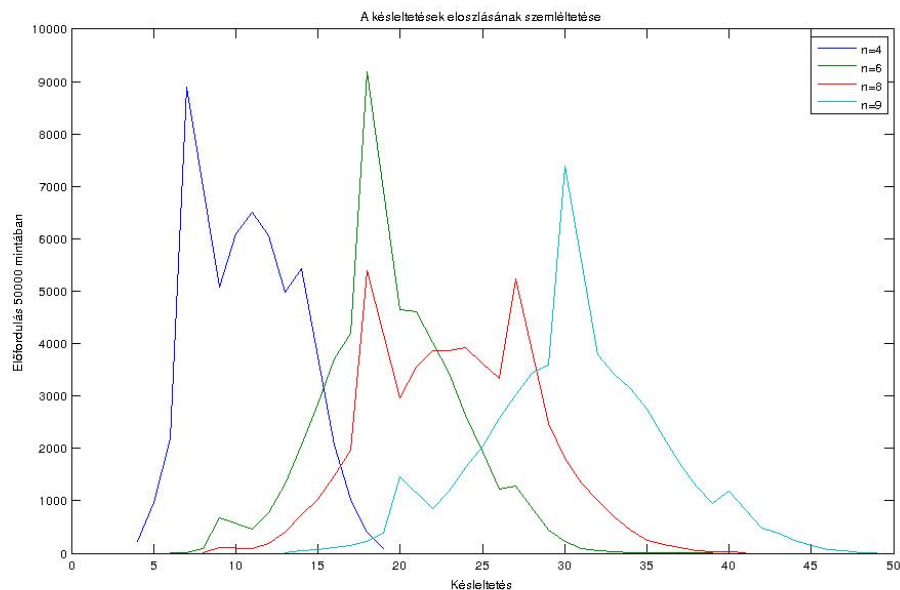
A késleltetések alakulását bemutató grafikon a 31. ábrán látható.



31. ábra: A késleltetések nagyságát bemutató grafikon. Kékkel jelölve a maximális, zölddel jelölve a minimális késleltetés látható.

A grafikonon a minimális késleltetés nem szimuláció, hanem számítás eredménye és csak a szemléltetés kedvéért szerepel a grafikonon. A minimális késleltetés akkor érhető el, ha az átlóban lévő csomópontokat tartalmazó csoportok úgy vannak ütemezve, hogy pont egymás után, a megfelelő sorrendben kerüljenek sorra.

A maximális késleltetés meghatározásához először az ütemező legenerálta a csoportokat, majd abból 500 véletlenszerű sorrend szimulálásra került. A grafikonra a legrosszabb eset került fel. Fontos azt is ellenőrizni, hogy a legrosszabb eset mennyire valószínű. Ennek vizsgálatára $n = 4, 6, 8, 9$ esetekben generálva lett egy-egy grafikon, mely a késleltetések eloszlását mutatja be a bal felső és jobb alsó csomópont között. A grafikon a 32. ábrán látható. Ez alapján levonható a következtetés, hogy az átlagos eset sokkal valószínűbb, mint a szélsőséges, n növelésével pedig nő a késleltetések várható értéke és szórása is.



32. ábra: A késleltetések eloszlása. A fenti grafikonon az látható, hogy $n = 4, 6, 8, 9$ esetén 50000 véletlenszerű sorrendben mekkora késleltetés hány esetben fordult elő a bal felső és a jobb alsó csomópont között.

5.4.2. Sor-topológia

A sor-topológia (melyben minden végpont csak a jobb és bal szomszédjával tud kommunikálni) egyszerű, de két ok miatt mégis érdemes vizsgálni:

- **gyakran előfordul**, például egy gyártósor mellett;
- könnyű fizikai eszközökkel is megvalósítani, így **validációs mérések** alapja lehet.

A szimulációk alapján egy sor-topológia mindig felosztható három, egymást nem zavaró kommunikációs csoportra. Ennek helyessége könnyen belátható: azok a csomópontok biztosan adhatnak egyszerre, amiknek nincs közös szomszédjuk. Annak feltétele, hogy két csomópontnak ne legyen közös szomszédja, akkor teljesül, ha legalább két másik csomópont van közöttük. Így a hálózatot három, egymással nem interferáló részre lehet osztható (33. ábra).



33. ábra: Sortopológia. A sortopológia mindig felosztható három, egymással nem interferáló csoportra. Az ábrán a csoportokat a színek jelképezik.

A késleltetés arányos a két csomópont távolságával, az arányossági szám pedig az ütemezés sorrendjétől függ. Bár a három csoportot hat lehetséges sorrendben lehet ütemezni, ez valójában csak két ténylegesen különböző ütemezést eredményeznek, a maradék négy ennek a kettőnek az eltoltja:

- az egyik lehetséges ütemezés az, hogy ha A ad az n . időrésben, akkor A bal szomszédja ad az $n + 1$ -ben. Ekkor a fő adatáramlás bal irányú;

- a másik ennek a tükörképe: ha A ad az n . időrásban, akkor A jobb szomszédja az az $n + 1$ -ben. Ekkor a fő adatáramlás jobb irányú.

Könnyen bebizonyítható (és a szimulátor is ezt az eredményt adja vissza), hogy ha a feladótól a címzett a fő adatáramlási irányban helyezkedik el és a közöttük lévő távolság D , akkor az üzenet D időrás alatt érkezik meg, egyébként $2 \cdot D$ időrás alatt.

A szimulációhoz szükséges validáció szintén sor topológiában történt meg. A mérés során a cél a szimulációs és mért eredmények összehasonlítása volt.

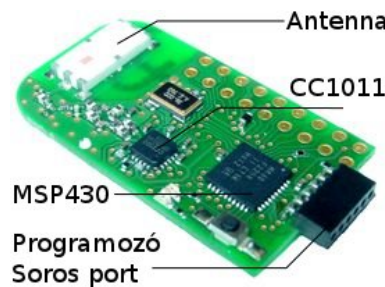
5.4.3. Validációs mérés

A validációs mérés 7 darab Amber 8423 demóboard segítségével történt (34. ábra).

Két féle program készült az eszközökre:

- a **producer** figyelte a gombját és amennyiben valaki megnyomta, felvillantott egy zöld LED-et és írt egy osztott paramétert;
- a **consumer-ek** az osztott paraméter alapján kapcsolgatták a saját LED-jeiket.

(A firmware mindkét csomópont esetén ugyan az volt, csak a rajtuk futó létrakód különbözött.)



34. ábra: Amber 8423 demóboard. A lapka mindent tartalmaz, ami egy egyszerű alkalmazáshoz szükséges.

A mérés során hat csomópont működött sor-topológiába rendezve. A sor elején állt a *producer*, utána a *consumer*-ek. A szimulációs adatoknak megfelelően amikor az adatáramlás iránya a producer-től a consumer-ek felé mutatott, a consumer-ek LED-jei egymás után kapcsolódtak fel. Amikor ellenkező irányban voltak, akkor két felkapcsolás között történt egy időrásnyi szünet.

Mivel a szimuláció és a mérés során kapott eredmények megegyeznek, ezért a szimulációt helyesnek lehet tekinteni.

A szimuláció feltételezett egy megbízható adatkapcsolati réteget, ennek hiányában számolni kell valamekkora valószínűségű csomagvesztéssel. A csomagvesztés nem volt része a szimulációnak, erre a protokoll nem is érzékeny, hiszen a producer-ek az általuk kezelt adatokat folyamatosan küldik. A későbbiekben érdemes elvégezni a szimulációkat különböző csomagvesztési scenáriókkal is, amire a szimulációs környezet már most is alkalmas.

6. Továbblépési lehetőségek

A dolgozatban bemutatam egy olyan rendszert, amely képes kiváltani a jelenleg elterjedt központosított vezérlést. A rendszer kifejlesztése során négy feladatott oldottam meg:

- olyan **végpontokat** terveztem és valósítottam meg, amelyek képesek futtatni egy összetettebb vezérlőlogikát;
- kifejlesztettem egy olyan **vezeték nélküli kommunikációs protokollt**, amely megfelel az ipari igényeknek;
- elkészítettem egy új **HTML alapú hálózat szimulátort**, amelyben kommunikációs protokollokat implementálva vizsgálható azok teljesítőképessége;
- létrehoztam egy olyan **fejlesztőkörnyezetet**, mely lehetővé teszi a fejlesztők számára akár több száz csomópont párhuzamos programozását is.

Bár a létrehozott rendszer a jelenlegi állapotában is használható, a sikerességéhez további fejlesztések szükségesek. Először a végpontok képességeit kell bővíteni különféle perifériákkal, mint például analóg ki- és bemenetekkel, impulzusjeladókkal és számlálókkal. Ezeket a speciális perifériákat a rugalmasság érdekében – hasonlóan a jelenleg elterjedt PLC-khez – érdemes modulárisra és bővíthetőre tervezni. A vezeték nélküli protokoll hibátűrését is növelni kell, hiszen a jelenlegi megvalósításában nem tudja kezelni a meghibásodott linkeket. A legegyszerűbb módszer erre az ütemező algoritmusának átalakítása úgy, hogy egyszerű részgráfok helyett Steiner-gráfokat keressen. Bár ez egy NP-teljes probléma, több algoritmus (például lineáris programozás) létezik a keresésükre. Azt is fontos jobban vizsgálni, hogy a kommunikációs csoportok sorrendjének létezik-e optimuma. Az éles használathoz a fejlesztőkörnyezet képességeit is bővíteni kell a kód szerkesztését és karbantartását egyszerűsítő eszközökkel.

A. Letölthető tartalom

A dolgozat során elkészült szoftverek megtalálhatóak a

<http://users.itk.ppke.hu/~lorma/otdk2015/>

weblapon. Az oldal pontokba szedve tartalmazza az elkészült firmware-t, a fejlesztőkörnyezet kipróbálható verzióját, a szimulációs környezetet és néhány előkészített szimulációt.

A weblapon található példák és programok összeválogatása során a cél az volt, hogy segítse a dolgozatban bemutatott algoritmusok, eszközök és rendszerek jobb megismerését azok kipróbálásának lehetővé tételével. (A firmware-t leszámítva minden kipróbálható a böngészőn keresztül.)

Hivatkozások

- [1] National Instruments (2014. 11. 13) Real-Time Industrial Controllers [Online] <http://sine.ni.com/nips/cds/view/p/lang/hu/nid/206845>
- [2] Texas Instruments (2014. 11. 13) Industrial System Solutions [Online] http://www.ti.com/lstds/ti/apps/industrial/overview.page?DCMP=hpa_contributed_article&HQS=industrial-ca
- [3] PLCDev (2014. 11. 13) The birth of PLC [Online] http://www.plcdev.com/the_birth_of_the_plc
- [4] ControlDesign (2014. 11. 13) Terminal/PLC Combo to Rise in Popularity [Online] <http://www.controldesign.com/industrynews/2012/terminal-plc-combo-to-rise-in-popularity/>
- [5] Automation (2014. 05. 03) Understanding Programmable Automation Controllers (PACs) in Industrial Automation [Online] http://www.automation.com/pdf_articles/Understanding_PACs_in_Industrial_Automation.pdf
- [6] Rockwell (2014. 05. 03) Micrologix system [Online] <http://ab.rockwellautomation.com/programmable-controllers/micrologix-systems>
- [7] Programmable controllers – Part 3: Programming languages (International Electrotechnical Commission, 2013) ISBN 9782832206614, Chapter 8 Graphic languages
- [8] PLCDev (2014. 11. 13) How PLCs work [Online] http://www.plcdev.com/how_plcs_work
- [9] Edward W. Kamen Industrial Controls and Manufacturing, (Academic Press, 1999) ISBN 0123948509, Chapter 8 Ladder Logic Diagrams and PLC Implementations
- [10] W. S. Levine The Control Handbook (CRC Press, 1996) ISBN 0849385709, Chapter 18. Ladder diagram
- [11] T. Moore (2014. 11. 13) Wlan and Bluetooth dominate market for wireless-enabled industrial automation equipment [Online] <http://eecatalog.com/wireless/2013/08/08/wlan-and-bluetooth-dominate-market-for-wireless-enabled-industrial-automation-equipment/>
- [12] D. Gislason, "ZigBee Is Highly Reliable" in Zigbee wireless networking
- [13] A. Van Den Bossche, "Proposition d'une nouvelle méthode d'accès déterministe pour un réseau personnel sans fil à fortes contraintes temporelles," Thèse, Université Toulouse le Mirail—Toulouse II, Toulouse, France, Jul. 6, 2007.
- [14] H. Monson. (1999. dec. 14.) Bluetooth Technology and Implications [Online]. Available: <http://www.sysopt.com/features/network/article.php/3532506>
- [15] H. Monson. (1999. dec. 14.) Bluetooth Technology and Implications [Online]. Available: http://www.sysopt.com/features/network/article.php/12029_3532506_2
- [16] ICTP (2014. 11. 13) Introduction to WiFi Networking [Online] http://www.piscespacific.org/livesite/files/Introduction_to_WiFi_Networking.pdf

- [17] mc mcse (2014. 11. 13) How Ethernet CSMA/CD Works [Online] <http://www.mcmcse.com/cisco/guides/csma.shtml>
- [18] Khaldoun A. A, Marc-Henry B, Tuan D, Alexandre G., Pascale M., Thierry V. and Jean-Baptiste V. "Which wireless technology for industrial wireless sensor networks? The Development of OCARI Technology" IEEE TRANSACTIONS OF INDUSTRIAL ELECTRONICS, VOL. 56, NO. 10, pp. 4266-4278, OCTOBER 2009
- [19] Libelium. (2012. május 8). About us [online]. Available: <http://www.libelium.com/company/about>
- [20] M. Lőrincz, "Sentinel box monitoring rendszer" Technika m, vol. 55, no. 2, pp. 26-27, Mar. 2012
- [21] Kan Y., Mikael G., Johan A., Math B., Reliable and Low latency transmission in industrial wireless sensor networks, The first international workshop on wireless network control systems, pp. 866-873.
- [22] L. Xue, X. Guen, Z. Liu, B. Yang (2014. 04. 12) TREE: Routing strategy with guarantee of QoS for industrial wireless sensor networks [Online] <http://onlinelibrary.wiley.com/doi/10.1002/dac.2376/abstract>
- [23] T. He, J. A Stankovic, C. Lu, T. Abdelzaher (2014. 04. 12) SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks [Online] http://www.cse.wustl.edu/~lu/papers/icdcs03_speed.pdf
- [24] R. Saravanakumar, N. Mohankumar, J. Raja (2014. 04. 12) Proficient Node Scheduling Protocol for Homogeneous and Heterogeneous Wireless Sensor Networks [Online] <http://www.hindawi.com/journals/ijdsn/2013/826482/>
- [25] Heinzelman, W., Chandrakasan, A., and Balakrishnan, H., "Energy-Efficient Communication Protocols for Wireless Microsensor Networks", Proceedings of the 33rd Hawaaiian International Conference on Systems Science (HICSS), January 2000
- [26] Liqun Hou and Neil W. Bergmann, "Novel Industrial Wireless Sensor Networks for Machine Condition Monitoring and Fault Diagnosis" IEEE TRANSACTIONS ON INSTRUMENTATION AND MEASUREMENT, VOL. 61, NO. 10, pp. 2787-2789, OCTOBER 2012
- [27] M. Wolf, Computers as components: principles of embedded computing system design. Elsevier, 2012.
- [28] 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
- [29] Béla G. Libták, Process Software and Digital Networks, (CRC Press, 2002) ISBN 0849310822, Chapter 4.4 ControlNet
- [30] P. Orponen (2014. 10. 14) The Steiner Network Problem [Online] http://www.tcs.hut.fi/Studies/T-79.7001/2008SPR/slides/orponen_080407.pdf
- [31] WiFi Virtual Laboratory (2014. 11. 13) Hidden node problem [Online] <http://wifi.cs.st-andrews.ac.uk/wifihiddennode.html>
- [32] Mathworks (2014. 05. 03) Low Energy Adaptive Clustering Hierarchy protocol (LEACH) [Online] <http://www.mathworks.in/matlabcentral/fileexchange/40115-low-energy-adaptive-clustering-hierarchy-protocol--leach->