Budapesti Műszaki és Gazdaságtudományi Egyetem
Villamosmérnöki és Informatikai Kar
Villamosmérnöki Szak
Mikrohullámú Híradástechnika Tanszék

Hálózati-, szállítási- és viszonyszintű protokollok implementációja MikroWAPszerver alkalmazáshoz

Dezső Tamás
konzulens: dr. Eged Bertalan
2001.

BUDAPESTI MŰSZAKI ÉS
GAZDASÁGTUDOMÁNYI EGYETEM
VILLAMOSMÉRNÖKI ÉS INFORMATIKAI KAR
MIKROHULLÁMÚ HÍRADÁSTECHNIKA TANSZÉK

DIPLOMATERV

Dezső Tamás
mérnökjelölt részére

Címe: **Hálózati-, szállítási- és viszonyszintű protokollok implementációja MikroWAPszerver alkalmazáshoz**

Feladat: Adjon áttekintést a hálózati szintű Internet Protocol (IP) felépítéséről és működéséről.
Ismertesse a szállítási réteg szintjén elhelyezkedő User Datagram Protocol-t (UDP).
Mutassa be a Wireless Session Protocol (WSP) MikroWAPszerver alkalmazáshoz szükséges részeit.
Valósítsa meg a fenti protokollok egyszerűsített változatát a MikroWAPszerver alkalmazáshoz.

Irodalom:

[1] WAP Forum: *Wireless Application Protocol Architecture Specification*, April 1998.
[2] WAP Forum: *Wireless Application Protocol Wireless Datagram Protocol Specification*, May 1999.
[3] WAP Forum: *Wireless Application Protocol Wireless Session Protocol Specification*, May 1999.
[4] Jon Postel: *RFC0791: Internet Protocol - DARPA Internet Program - Protocol Specification*, USC/Information Sciences Institute, 1981
[5] Jon Postel: *RFC0768: User Datagram Protocol*, Information Sciences Institute, 1980

Főszakirány: Híradástechnika
Záróvizsga tárgyak:

	A tárgy neve	kódja
1	Műholdas és mobil kommunikáció	VIMH 5010
2	Interfésztechnika	VIAU 4101
3	Szoftvereszközök	VIAU 4102

A feladat benyújtásának határideje: 2001. május 18.

Tanszéki konzulens: dr. Eged Bertalan

Ipari konzulens –

Tervezés bírálója:

Budapest, 2001.

.....
Dr. Zombory László
egyetemi tanár, tanszékvezető

Ezen nyomtatvány a diplomatervhez csatolandó

Konzultációk:

Időpont év	hó	nap	Észrevételek	Konzultáció

Tanszéki konzulens véleménye:

Bíráló véleménye:

Nyilatkozat

Alulírott, *Dezső Tamás*, a Budapesti Műszaki és Gazdaságtudományi Egyetem hallgatója kijelentem, hogy ezt a diplomatervet meg nem engedett segítség nélkül, saját magam készítettem, és a diplomatervben csak a megadott forrásokat használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

.....

Dezső Tamás

Kivonat

A diplomaterv keretein belül sor került egy mikrokontrollerre átültethető, teljes funkcionalitású, távoli hardvervezérlésre és lekérdezésre is alkalmas WAP (Wireless Application Protocol) kiszolgáló hálózati, szállítási, viszony- és alkalmazási rétegeinek tervezésére, implementálására és vizsgálatára. A munka részét képezte az Internet és a WAP réteges felépítésű protokollrendszerének tanulmányozása és a MikroWAPszerver alkalmazáshoz felhasználandó, minimális erőforrásigényt támasztó rétegek és az azok közötti interfészek részletes kidolgozása és megvalósítása. Az elkészült mikrokiszolgáló képes teljes értékű kommunikációra az arra megfelelő mobiltelefonnal, mint klienssel, és a mikrokontrollerre történő átültetés után, kliens- és szerveroldali mobilitást biztosítva, alkalmazható WAP felületen keresztüli vezérlési feladatok ellátására.

Tartalomjegyzék

1. fejezet

Bevezetés

Minden nap használjuk a technika legújabb vívmányait, és egyre növekszik az igény, hogy a mozgásszabadság korlátozása nélkül tehessük meg mindezt. Ennek az igénynek a kielégítésére születtek meg a hordozható számítógépek és a mobiltelefonok. Korunk két nagy ütemben fejlődő technikája, a vezetéknélküli adatelérés és az Internet konvergenciájának eredményeként jött létre az Internet egyes alkalmazásait mobiltelefonon elérhetővé tevő WAP (Wireless Application Protocol) [1]. Az Internet legnépszerűbb szolgáltatása, a World-Wide Web (WWW) jelentős fejlődésen ment keresztül megjelenése óta. Napjaink WWW szerverei rengeteg lehetőséget kínálnak, beleértve például a dinamikus, adatbázis alapú tartalom-generálást is. Ilyen kiszolgálók alkalmazásával megvalósítható a hardvervezérlés funkció is, amelynek segítségével bárhonnan a világon – egy internetkapcsolattal rendelkező számítógépről – vezérelni, illetve lekérdezni lehet olyan eszközöket, amelyek valamilyen formában (a kiszolgáló) számítógéphez csatlakoztathatók. Ha az internetelérést biztosító számítógépet egy WAP-böngészésre alkalmas mobiltelefonnal helyettesítjük, akkor az előzőekben bemutatott lehetőséget mobilan, bárhonnan elérhetjük. A kliensoldali mobilitás adta szabadság tovább fokozható, ha a szerver sincs helyhez kötve. Hagyományos, számítógépes megoldások esetében ez elképzelhetetlen, ugyanis egy internetszerver nagy helyet foglal el és folyamatos internet-hozzáférés nem létesíthető bárhol. Ezekre a problémákra ad megoldást a diplomatervezés során elkészített MikroWAPszerver.

A világ sok laboratóriumában fejlesztenek kisméretű és ugyanakkor nagytudású beágyazott rendszereket. A mikrokiszolgáló területen is nagy számban születtek, a kereskedelmi forgalomban is kapható készülékek. Léteznek olyan mikroszerverek, amelyek architektúrája és teljesítménye egy középkategóriájú személyi számítógépével azonos. Ilyen felépítésű eszközökön operációs rendszerként könnyen felhasználható bármelyik Linux disztribúció váza. A mikrokiszolgálók ezen típusánál valójában csak a mikroelektronika lehetőségeit használják ki, és alkalmazzák a már létező implementációkat. Más megközelítése a problémának, amikor egy programozható áramkör felhasználásával, és az adott architektúrára készült és optimalizált program alkalmazásával valósítják meg a

miniatűrizált kiszolgálót. A diplomatervezés keretein belül megvalósított, kliens- és szerveroldali mobilitást biztosító kiszolgáló, a MikroWAPszerver is ezen elv alapján készült.

1.1. A diplomaterv felépítése

2. fejezet: Hasonló témakörben készült munkák bemutatása, a diplomatervezési feladat értelmezése és részletes elemzése.

3. fejezet: Általános áttekintés a protokollrendszerekről és a MikroWAPszerver megvalósításához felhasznált protokollok architektúrájáról és működéséről.

4. fejezet: A szoftverfejlesztés egyes fázisainak nyomon követése és a fejlesztés során előállt specifikáció és implementáció részletes ismertetése.

5. fejezet: Az elkészült mikrokiszolgáló képességeit és hatékonyságát elemző mérések, vizsgálatok és azok eredményeinek ismertetése.

7. fejezet: A diplomatervezés során elvégzett feladatok és a vizsgálatok eredményeinek összefoglalása.

6. fejezet: A MikroWAPszerverrel kapcsolatos jövőbeli tervek és továbbfejlesztési lehetőségek bemutatása.

2. fejezet

Miniatűr kiszolgálók és a MikroWAPszerver

2.1. Mikrokiszolgálók

Számos laboratóriumban folynak fejlesztések miniatűr kiszolgálókkal kapcsolatban, elsősorban azonban a webszerverek témaköre örvend nagy népszerűségnek, az újnak mondható WAP terület ebben a tekintetben teljesen érintetlen. A kereskedelemben is kapható, szabadalmaztatott, illetve szabad forrású kiszolgálóknak több típusa is van. A legkézenfekvőbb megoldás egy számítógépet miniatűrizálni, felinstallálni rá egy operációs rendszert és feltelepíteni a hálózati kommunikációt lehetővé tevő protokollokat. Ezt a megoldást a Stanford Egyetemen egy gyufásdoboz nagyságú áramkörben realizálták [2]. A hardver lényegében egy 486-os számítógép, AMD 486-SX, 66MHz-es processzorral, 16MByte operatív memóriával és 16MByte háttértárral, amin egy RedHat Linux 5.2-es operációs rendszer fut egy WWW szerverrel. A külvilággal két soros porton, egy párhuzamos porton és egy floppy csatlakozón tud kommunikálni. A teljesítményt tekintve nyilvánvalóan ez a leghatékonyabb megoldás, mérete is elég kicsi ahhoz, hogy bárhol elférjen és bármikor mozdítható legyen. Egy ilyen kis méretben megvalósított számítógép, számos előnye mellett azonban igen drága.

Több, lényegében teljesen azonos elvű implementáció is létezik, amelyek méretét tekintve egy soros portra illeszthető csatlakozóban is elférnek, így tehát soros interfészen kommunikálnak [3, 4, 5]. Ezek működtetéséhez természetesen számítógép is szükséges, tehát a rendkívül kis méretük ellenére nagy hátrányuk, hogy csak számítógéppel együtt alkalmazhatók. Ahol azonban rendelkezésre áll egy számítógép, ahelyett hogy egy kis alkatrészbe kellene zsúfolni az egész protokollrendszert, az a számítógépen is megoldható lenne.

A mikrokiszolgálók fejlettebb kategóriája képes teljesen önálló működésre más számítógép, vagy egyéb eszköz nélkül is [6, 7, 8]. Az ide tartozó szerverek általában Ethernet

hálózati csatolón keresztül érhetők el és mikrokontroller alapúak. Erőforrásaik közel azonosak, általánosan néhány MHz-es processzorral, néhány KB-os operatív memóriával és a program tárolására alkalmas nagyságú flash ROM-mal rendelkeznek. Tipikusan körülbelül egy hitelkártya fele nagyságúak. A méretet, a mobilitást, a funkcionalitást és az árat tekintve ezek az eszközök jelentik a legoptimálisabb megoldást.

Az előbbi két megoldás hasznos tulajdonságait ötvözi az egyik legkisebb méretű és legnagyobb tudású eszköz [9, 10]. Ez egy olyan mikrokiszolgáló, amely önmagában soros interfészen keresztül történő kommunikációra képes, de egy, a méretét növelő kiegészítő egységgel Ethernet hálózatra is illeszthető. Egyetlen alapkiszolgálóval tehát kétféle alkalmazási igény is kielégíthető.

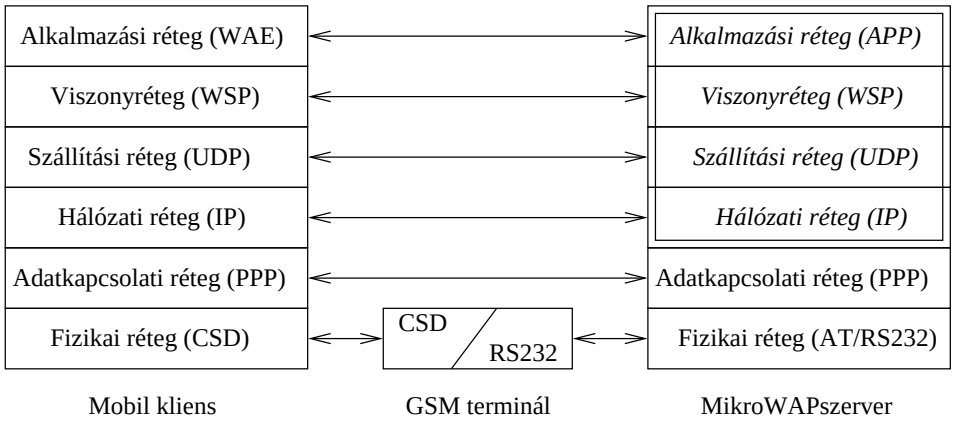
A bemutatottak alapján elmondható, hogy többféle elv alapján fejlesztett mikrokiszolgálók léteznek. Vannak egészen kisméretű és kisebb képességű-, vannak önállóan működőképes, nagytudású- és léteznek teljesen számítógép alapú, gyufásdoboznyi eszközök is. Előnyük a kis méret mellet, egy számítógép költségeihez viszonyított meglehetősen alacsony áruk. Igazán gazdaságos megoldást azok a szerverek jelentenek, amelyek számítógép nélkül is képesek ellátni feladatukat. A kínálatból könnyedén választható a megoldandó feladathoz legjobban illeszkedő realizáció, azonban a bemutatott eszközök egyike sem elégíti ki a kliensoldali teljes mobilitás igényét. A kiszolgáló viszonylag könnyen mozdítható kis mérete miatt, azonban a hálózati csatlakozási pont, illetve a számítógépes kapcsolat a szerveroldalon is korlátoz a mobilitásban, kliensoldalról pedig csak internetcsatlakozással rendelkező számítógépről érhető el. A diplomatervezés tárgyát képező MikroWAPszerver elve egyszerre kínálja fel a szerver és a kliens mobilitását, ugyanis mindkét oldalon az egyetlen követelmény a kapcsolat fizikai rétegét biztosító GSM terminál hálózati lefedettsége.

2.2. A MikroWAPszerver

A MikroWAPszerver egy olyan kisméretű, mobil eszköz, amely nem igényel számítógépet, és internetkapcsolatot a WAP felületen keresztül megvalósítható, bárhol[1] alkalmazható hardvervezérlés, illetve távlekérdezés funkció kiváltásához. Az előzőek értelmében a kiszolgáló három logikai egységből épül fel: egy GSM (Global System for Mobile Communications) terminálból, az ehhez csatlakozó, a protokollrendszert realizáló mikrokontroller egységből, valamint a külső eszközt a kiszolgálóhoz illesztő interfészből.

A WAP tartalomszolgáltató és alkalmazás-kiszolgáló funkció implementálásához, valamint a mikrokontrollerre való átültetés lehetővé tételéhez elengedhetetlen a kommunikáció során használatos protokollok és a felhasználandó mikrokontroller erőforrásainak részletes megismerése. A tervezésre csak a lehetőségek, és a követelmények pontos is-

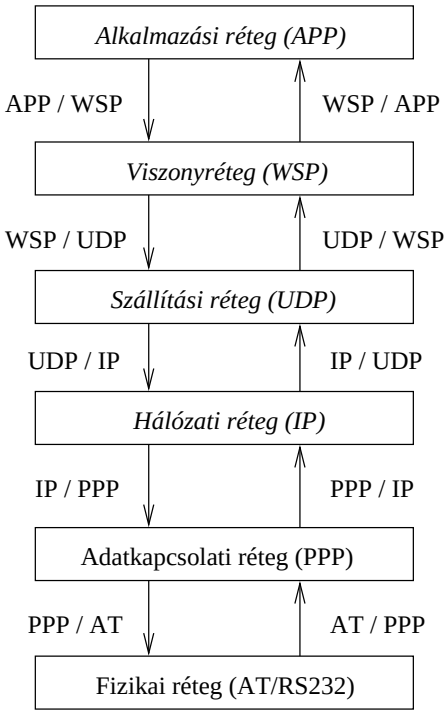
[1] ahol a működéshez szükséges minimális energia rendelkezésre áll akkumulátor, telep, vagy hálózati áramforrás formájában



2.1 ábra
A MikroWAPszerver alkalmazása során használt eszközök, rétegek és protokollok

meretében kerülhet sor. A tervezés során tehát folyamatosan szem előtt tartandó, hogy minimális erőforrás áll rendelkezésre, és az elkészítendő protokollrendszer képes legyen a kommunikációs út másik végén elhelyezkedő végberendezés protokolljaival együttműködni. A megoldás során feltételezzük, hogy az adatkapcsolati szintet is beleértve az alsóbb rétegek rendelkezésre állnak, és a hálózati réteg felé mutatott interfész a szabványnak megfelelő. A feladat tehát a hálózati és a felette elhelyezkedő szállítási, viszony- és alkalmazási rétegek, illetve azok protokolljainak megismerése, tervezése és megvalósítása az adott architektúrára optimalizálva. A kommunikáció során használt protokollok a résztvevő eszközök szintjére lebontva, valamint a MikroWAPszerver jelen diplomaterv keretein belül megvalósítandó rétegei (dőlt betűvel) a 2.1 ábrán vannak feltüntetve. Az ábra jól szemlélteti azt is, hogy hogyan kapcsolódik a három egység egymáshoz: A WAP böngészővel ellátott mobiltelefon (kliens) adathívás (CSD – Circuit-Switched Data) útján felépíti a kapcsolatot – az egyébként teljes funkcionalitású, de kezelőszerveitől megfosztott – GSM mobil terminállal. A terminál a soros interfészén keresztül van csatlakoztatva a MikroWAPszerverhez, így tulajdonképpen a kliens a mobil terminálon, mint közvetítőn keresztül kommunikál a mikrokiszolgálóval. A GSM terminál egyetlen feladata tehát, hogy a szerver soros interfészére illessze a mobil klienst.

Az egyes rétegek között biztosítani kell az adattovábbítást, amelynek eléréséhez üzenetvárakozási sor (queue) alkalmazható. A rétegszerkezet legalján és legtetején elhelyezkedőeket kivéve, minden egyes réteghez négy darab üzenetküldési sor rendelhető, amin keresztül küldik, illetve fogadják az üzeneteket. A sor tehát nem más, mint a szolgálatkérés, illetve teljesítés jelzésének eszköze. Az egy réteghez kapcsolódó négy üzenetküldési sor közül kettő magához a réteghez, a másik kettő pedig annak szomszédjaihoz vezet. Természetesen a protokollrendszer alján és csúcsán elhelyezkedő fizikai, illetve alkalmazási rétegeknek csak egy-egy szomszédjuk van, így hozzájuk összesen két-két queue kapcsolódik. A 2.2 ábra az egyes rétegek közötti interfészeket, illetve az üzenetküldési



2.2 ábra A MikroWAPszerver rétegei, protokolljai és interfészei

sorokat szemlélteti. A megvalósítandó rétegek (hálózati, illetve a felette elhelyezkedők) végül együtt fognak működni az alsóbb rétegekkel (adatkapcsolati, fizikai) a mikrokontrolleren, ezért az egyes rétegek közötti kommunikáció biztosítására célszerű egységes üzenet-várakozási sort alkalmazni.

Az előzőek, illetve a mikrokontroller adottságainak figyelembevételével, valamint a felsorolt protokollok felépítésének és működésének ismeretében kerülhet sor a MikroWAPszerver protokolljainak specifikálására és implementálására. A specifikáció során szem előtt tartandó, hogy a mikrokiszolgáló nem egy hálózathoz[2] csatlakozik, hanem a GSM terminál közvetítésével (soros vonalon keresztül) pont-pont kapcsolatban van a kliens mobiltelefonnal. Ilyen esetben a hálózati réteg feladata valamelyest leegyszerűsödik. A tervezésnél ugyancsak figyelembe veendő, hogy a kommunikációs út – a pont-pont kapcsolat miatt – eléggé megbízható, valamint a WAP szabvány [11] előírja a viszonyréteg nem kapcsolatorientált metódusának implementálását a WAP képes mobiltelefonok gyártói számára. Így alkalmazható a viszonyréteg nem kapcsolatorientált szolgáltatása a kommunikáció során, azaz nem feltétlenül szükséges a WSP (Wireless Session Protocol) [11] kapcsolatorientált metódusának megvalósítása a kiszolgáló oldalon. Ezzel is csökkenthető a célhardver felé támasztott mind tárterület, mind processzoridő formájában jelentkező erőforrásigény.

A diplomatervezés eredményeként elkészült implementációt célszerű több tekintetben is megvizsgálni, tesztelni, elemezni. Ezen vizsgálatok eredményei alapján ítélhető meg,

[2]Ethernet, Tokenring, vagy egyéb típusú helyi hálózat

hogy a megvalósított kiszolgáló megfelel-e a specifikációnak és a különböző funkcionalitási, hatékonysági, biztonsági és stabilitási elvárásoknak.

A feladat tehát az előzőekben ismertetettek figyelembevételével megtervezni és implementálni a MikroWAPszerver hálózati és a felette elhelyezkedő rétegeit. A megvalósításhoz a C programozási nyelv használata célszerű, ugyanis a modern mikrokontrollerek rendelkeznek standard C fejlesztői környezettel és fordítóval, a fejlesztés pedig így egyszerűbb, mint az alacsonyszintű és sokkal inkább hardver-specifikus assembly-t használva. Az elkészült implementációt számítógépes környezetben, mikrokontrollerre történő átültetés nélkül is lehet vizsgálni, tesztelni, de az eredmények értékelésekor természetesen figyelembe kell venni az architektúrák közötti különbségeket.

3. fejezet

Protokoll-rendszerek

A MikroWAPszerver alkalmazása során nem beszélhetünk valódi hálózatról, ugyanis a kliens és a szerver között pont-pont kapcsolat van. Az egyes alkalmazások, mint például a World-Wide Web, valójában hálózati alkalmazások, amelyek működése meghatározott protokollrendszerre épül. Abban az esetben, ha ugyanazt a hálózati alkalmazást kívánjuk pont-pont kapcsolat esetén használni, a kommunikáció elveszíti a hálózati jellegét, de ugyanazt a protokollrendszert használja, mint egyébként. Ezért szükséges a hálózati architektúrák, illetve protokollrendszerek áttekintése.

3.1. Hálózati architektúrák

A napjainkban használatos számítógép-hálózatok a strukturált tervezés eredményei. Az egyes jól meghatározott funkciókat egymástól elkülönített egységekben realizálják úgy, hogy közben biztosítják a részegységek közötti kommunikációt.

3.1.1. Protokoll-hierarchiák

A számítógép-hálózatokat a meghatározott feladatok különválasztása, valamint a tervezés strukturálása miatt rétegekre osztották [12]. Ezek úgy épülnek egymásra, hogy a felsőbb réteg az alatta elhelyezkedő szolgáltatásait használja, illetve az alsóbb réteg szolgáltatásokat nyújt a felette levőnek. A rétegek darabszáma, valamint azok funkciója hálózatonként más és más. Általános funkció a felsőbb réteg számára szolgáltatást biztosítani, azaz eltakarni előle a szolgáltatás megvalósítását. A kommunikáció a kapcsolatban levő gépek azonos szinten elhelyezkedő rétegei között zajlik. A kommunikáció szabályait leíró specifikáció a protokoll. Fizikailag azonban nem az egyik gép egyik rétege küld adatot a másik gép azonos szintű rétegének, hanem azok – egy-egy gépen belül – küldik az adatokat és a vezérlőinformációkat egymásnak. Egy adatküldés alkalmával az adat végighalad az egész strukturált rendszeren és a legalsó (a fizikai) rétegen keresztül jut majd csak el a célgépig. A célgépnél az adat a fizikai rétegtől üzenetátadások útján

kerül át a forrásgép által megcímzett réteghez. A rétegeket a közöttük húzódó interfész kapcsolja össze. Az interfész a felsőbb rétegnek nyújtott szolgáltatásokat definiálja. Az interfészek tisztasága alapvető követelménye a hálózatnak, ezért ez fontos szerepet játszik a tervezésnél, valamint a protokollok számának és funkciójának meghatározásakor. A jól definiált, tiszta interfészek és a réteges szerkezet hozzájárulnak ahhoz, hogy egy-egy réteg szükséges esetben kicserélhető legyen egy hasonló funkcionalitásúra. A rétegeket és a rétegek közötti kommunikációt szabályrendszerbe foglaló protokollokat együttesen hálózati architektúrának nevezzük. Az implementáció részletei, az interfészek és azok azonossága nincs meghatározva az architektúrában, hiszen ezek a gépek belsejében vannak, és kívülről nem láthatók.

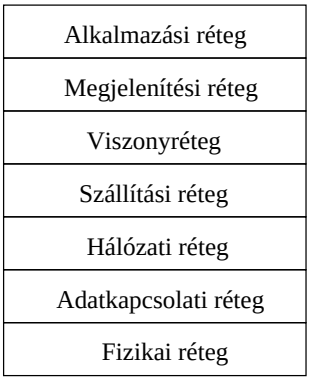
A többszintű kommunikáció többet jelent az adatok egyszerű továbbításánál, bár a cél, illetve a végeredmény nem több ennél. A rétegek a továbbítandó adatokkal együtt úgynevezett vezérlőinformációkat is küldenek. Ezen járulékos információk alapján döntik el az egyes rétegek, hogy milyen műveleteket kell végrehajtani az adatcsomagokon. A vezérlőinformációk általában az adat elé illesztett fejlécben *(header)* továbbítódnak. Ha egy réteg fejlécet fűz az adatokhoz, akkor a célgép megfelelő rétegéhez képest magasabb szintű rétegek nem kapják meg a járulékos információkat, azok a küldővel azonos szinten levő rétegnek szólnak.

Azért szükséges a feladatok rétegekbe sorolása tehát, hogy az eredetileg szinte kezelhetetlen rendszer tervezését kisebb, jól definiálható részekre lehessen osztani.

3.1.2. Rétegek tervezési kérdései

A tervezés során felmerülő kérdések, illetve problémák, a rétegekre osztásnak köszönhetően nem a teljes hálózati architektúra, hanem rétegek tervezési kérdéseivé válnak, így egy-egy problémát elég az adott réteg szintjén megoldani [13].

A rétegeknek rendelkeznie kell a kapcsolat felépítését, illetve lebontását végrehajtó mechanizmusokkal. Egy hálózat több gépből áll ezért lehetőséget kell biztosítani a kapcsolat felvételére a hálózat egy kijelölt gépével, valamint amikor a kapcsolat feleslegessé válik, annak lebontása is szükséges. Az előző okból kifolyólag szükség van címzésre is, amellyel meghatározhatjuk, hogy melyik géppel kívánunk kapcsolatot létesíteni. Fontos kérdés lehet az is, hogy milyen irányú, illetve szervezésű legyen az adatátvitel. Ha az adat csak egy irányban haladhat, szimplex, ha egyszerre csak egy irányban, de felváltva két irányban, akkor fél-duplex, ha mindkét irányban egyidejűleg, akkor duplex kommunikációról beszélünk. Egy hálózati összeköttetés lehet bizonytalan is, ezért gondolni kell a hibajavításra. Ez megoldható hibajavító kódolással, illetve az adatok (helyes) megérkezését igazoló nyugtázással. Előfordulhat, hogy egy réteg nem fogadhat egy meghatározott méretnél nagyobb csomagokat, így az eredeti adategység darabolásra, kisebb csomagokra osztásra kerül. A szétdarabolást követően a részcsomagokat a másik oldalon össze is kell



3.1 ábra OSI referencia modell

illeszteni, ezért azokat valamilyen rendszer szerint sorszámozni kell. Egy nagyobb hálózaton nem feltétlenül biztosítható, hogy a csomagok a célhoz a kiküldés sorrendjében érkeznek meg, ezért gondoskodni kell a sorrendhelyességről is. Az üzeneteknek valamilyen úton el kell jutni a célállomáshoz, és ehhez rendszerint valamilyen útvonal-választási eljárás szükséges. Ha egy réteg szolgáltatásait több felsőbb réteg is igénybe szeretné venni egyidejűleg, akkor ez a multiplexálás, illetve a nyalábolás technikájával oldható meg. Számos döntési helyzet áll tehát elő már akár egyetlen réteg tervezése során is.

3.2. Az OSI referencia modell

A Nemzetközi Szabványügyi Szervezet (ISO – International Standard Organization) 1983-ra kidolgozott egy ajánlást a nyílt rendszerek összekapcsolására vonatkozóan. Erre az ajánlásra épül a 3.1 ábrán látható OSI (Open System Interconnect – Nyílt Rendszerek Összekapcsolása) [12] hivatkozási modell. Az ISO OSI modell hét rétegből áll. Egyes, például az Interneten is használt protokollrendszerek ennél kevesebb réteget definiálnak, felmerülhet a kérdés, hogy miért pont hét rétegű ez a modell. A modell kialakításakor és a rétegek számának meghatározásakor több szempont is mérvadó volt. Fontos volt, hogy az egyes rétegek különböző absztrakciós szinteket képviseljenek, minden rétegnek jól definiált feladata legyen, a réteghatárok úgy legyenek megválasztva, hogy az interfészeken minél kevesebb járulékos információt kelljen továbbítani, megfelelően nagyszámú réteg legyen ahhoz, hogy különböző jellegű feladatok ne kerüljenek egy rétegbe, valamint eléggé kis számú réteg legyen ahhoz, hogy a rendszer szerkezete ne váljon nehezen kezelhetővé, érthetetlenné. Megjegyzendő azonban, hogy az OSI hivatkozási modell nem egy hálózati architektúra, mivel a szabvány nem határoz meg hozzá protokollokat és szolgálatokat.

3.2.1. Fizikai réteg

A fizikai réteg (physical layer) felelős a bitek továbbításáért. Biztosítania kell az adó oldalon kibocsátott bitek megérkezését és helyes értelmezését a vevő oldalon. Jelen réteg hatáskörébe tartozik definiálni, hogy mekkora feszültségszintek, illetve időintervallumok feleljenek meg a logikai „0”, illetve „1” értékeknek; történhet-e egyidejű – kétirányú adatátvitel; hogyan épüljön fel, illetve bomoljon le a kapcsolat; milyen típusú legyen, milyen mechanikai és villamos tulajdonságokkal rendelkezzen a hálózati csatlakozó.

3.2.2. Adatkapcsolati réteg

Az adatkapcsolati réteg (data link layer) feladata, hogy bármely kezdetleges adatátviteli eszközt hibamentes átviteli csatornává transzformáljon a hálózati réteg számára. Ez úgy érhető el, ha az adatot meghatározott méretű keretekbe foglalja, azaz bizonyos bitszekvenciákat helyez az adatblokkok elé és mögé. Mivel a fizikai réteg feladata csak a bitek továbbítása, az nem tud a keretekről sem, ezért a keretinformációk beillesztése, valamint azok felismerése is az adatkapcsolati réteg feladata. A keretek megérkezéséről – nyugtakeretek felhasználásával – tájékoztatni kell a küldő felet. Ha egy keretre nem érkezik nyugtakeret meghatározott időn belül, akkor azt újra kell küldeni. Előfordulhat olyan eset is, hogy a nyugtakeret nem érkezik meg valamilyen külső hatás, például nagymértékű zaj, villámcsapás stb. következtében, ekkor a küldő fél úgy érzékeli, hogy nem küldtek nyugtát az előzőekben küldött keretről, így azt újra kell küldeni. Ilyen esetben a kérdéses keret kétszer érkezik meg a címzetthez, amelynek ezt a problémát kezelnie kell, tehát a két keret közül csak az egyiket kell feldolgoznia. Nem csak az adatkapcsolati, hanem más rétegek problémája is az adatelárasztás, ami egy gyors és egy lassú gép közötti kommunikáció esetén léphet fel. A küldő olyan gyorsan küldi a kereteket, hogy a vevő a buffere megtelése után már nem tudja fogadni azokat. Megoldás erre a problémára egy forgalomszabályozási mechanizmus, amely során az adót folyamatosan tájékoztatjuk a vevő szabad bufferméretéről. Problémát okozhat, ha vonal kétirányú forgalomra is használható, hogy az adatkeret-forgalom versenyez a nyugtakeret-forgalommal a csatorna használati jogáért.

3.2.3. Hálózati réteg

A hálózati réteg (network layer) feladata a kommunikációs alhálózatok működésének vezérlése. Igen fontos kérdése a hálózati réteg tervezésének az útvonal-választás. A forrás és a cél között az útvonal többféleképpen is kijelölhető. Kisebb hálózatokban ez történhet rögzített táblázatok alapján, teljesen fix módon. Alkalmazható útválasztás úgy is, hogy a kommunikáció kezdetén rögzítjük az útvonalakat és a kommunikáció folyamán végig ezt használjuk, illetve történhet teljesen dinamikusan is, minden egyes adatcso-

mag elküldése során. A legutóbbinak az a nagy előnye, hogy minden egyes pillanatban figyelembe vehető a hálózat terheltsége, és ettől függően választhatunk kevésbé terhelt útvonalat is. Torlódás alakulhat ki abban az esetben, ha az adott alhálózatban túl sok csomag van egyszerre jelen. Az ilyen jellegű problémák kezelése is a hálózati réteg feladata. Mivel bizonyos hálózatok üzemeltetői nem ingyenesen kínálják szolgáltatásaikat, a hálózati rétegnek lehetőséget kell biztosítania valamilyen – például az átvitt bitek számán alapuló – számlázásra. Ha egy csomag több alhálózaton keresztülhaladva jut el a célállomásig, problémát okozhatnak az egyes alhálózatok különböző címzési módszerei, illetve az azokon megengedett maximális csomagméretek. Ezen nehézségek áthidalásáért, azaz a heterogén rendszerek összekapcsolásáért is a hálózati réteg felel.

3.2.4. Szállítási réteg

A szállítási réteg (transport layer) által elvégzendő feladat, hogy fogadja a viszonyrétegtől az adatokat, majd az esetleges szétdarabolás után továbbítsa a hálózati rétegnek úgy, hogy közben azt is biztosítja, hogy minden csomag hibátlanul érkezik meg a másik oldalra. Ezt a feladatot úgy kell ellátnia, hogy a viszonyréteg elől eltakarja a hardvertechnikai különbségeket (például bájtsorrend). A szállítási réteg általában minden egyes, a viszonyréteg által kért szállítási kapcsolat számára új hálózati összeköttetést hoz létre. Különös esetben előfordulhat, hogy az egy hálózati összeköttetés által biztosított adattovábbítási sebesség nem elegendő a viszonyréteg által küldött adatok továbbítására. Ilyenkor a szállítási réteg megnyithat több hálózati kapcsolatot is, nagyobb áteresztőképességet biztosítva így a viszonyrétegnek. Előfordulhat az is, hogy például a magas költségek miatt nemhogy több hálózati kapcsolatot nyit egyetlen viszony kiszolgálásához, hanem több szállítási kapcsolatot nyalábol egyetlen hálózati összeköttetésre. Ezeknek a műveleteknek a viszonyréteg számára teljesen átlátszónak kell lennie. A szállítási összeköttetéseknek számos típusa van. Ezek közül a legjobbnak és legszívesebben használtnak mondható a két pont közötti hibamentes csatorna, amelyen még az üzenetek sorrendhelyessége is biztosított. Létezik olyan eset is, amikor az üzenetek sorrendhelyessége nem követelmény így a szállítási réteg nem is biztosítja azt. Az előzőektől teljesen különböző az a típusú szolgáltatás, amikor az üzenetet nem egy meghatározott gépnek, hanem egy csoportnak küldik.

A szállítási szint alatti rétegek protokolljai valójában nem a forrás- és a célgép közötti kommunikációt valósítják meg, hanem a szomszédos gépeken keresztül kommunikálnak. Ezekkel ellentétben a szállítási réteg valódi két végpont közötti réteg. A forrásgép szállítási rétege fejlécekben továbbított vezérlőinformációk felhasználásával kommunikál a célgép azonos rétegével a szállítási protokollt felhasználva a kommunikáció lebonyolítására.

A szállítási rétegnek biztosítania kell azt is, hogy azonosítani lehessen, hogy me-

lyik üzenet melyik összeköttetéshez tartozik, ugyanis általában lehetőség van arra, hogy egy gépen több összeköttetés is éljen egyidejűleg. Szükség van egy névadási mechanizmusra is, amelynek felhasználásával a forrás azonosíthatja a célgépnek azt a folyamatát, amellyel kommunikálni akar.

A szállítási rétegnek az alsóbb rétegeknél is említett problémát, a gyors gép által a lassabbon okozott elárasztást is kezelnie kell. Szállítási szinten is szükség van tehát egy olyan eljárásra, amellyel az adatáramlást szabályozni lehet.

3.2.5. Viszonyréteg

A viszonyréteg (session layer) azt biztosítja, hogy a kapcsolatban álló gépek egymással felhasználói viszonyt létesíthessenek. Az egyszerű adatátvitelt ez a réteg is kiegészíti az egyes alkalmazásokhoz szükséges járulékos szolgáltatásokkal. Egy viszony lényegében azt teszi lehetővé, hogy egy felhasználó bejelentkezhessen egy másik rendszerbe, illetve adatokat (állományokat) továbbíthasson. Ezen funkciók elvégzéséhez nyújt szolgálatot a viszonyréteg kölcsönhatás-menedzselés, illetve szinkronizáció formájában.

A kölcsönhatás menedzselés annyit jelent, hogy a kommunikáció során biztosítani lehet azt, hogy a két oldal ne próbálkozzon ugyanannak a műveletnek az egyidejű végrehajtásával. Ezt például egymásnak átadható vezérjelek alkalmazásával lehet megoldani. Az adott művelet végrehajtására az jogosult, aki a vezérjelet birtokolja. A protokollnak természetesen szabályoznia kell azt, hogy a vezérjelet melyik fél és mennyi ideig, illetve hány művelet erejéig birtokolhatja.

A szinkronizáció arra alkalmas, hogy egy folyamatba bizonyos meghatározott helyeken szinkronizációs pontokat iktasson, és a folyamatot bármikor folytatni lehessen a legutolsó szinkronizációs ponttól. Ennek felhasználásával megakadályozható például az, hogy egy igen hosszú állományletöltés a kapcsolat megszakadása után teljesen elölről kezdődjön. Ehelyett a megszakadást megelőző utolsó szinkronizációs ponttól folytatható a művelet.

3.2.6. Megjelenítési réteg

A megjelenítési réteg (presentation layer) feladatköre már túlmutat az adatok biztonságos továbbításán. A megjelenítési réteg az átviendő információ szintaktikájával és szemantikájával foglalkozik. Legfontosabb feladata az adatok szabványos kódolása. A felhasználói programok nem véletlen bitsorozatokat akarnak továbbítani, hanem valamilyen struktúrába szervezett adatokat, mint például számokat, karakterfüzéreket, dátumokat stb. Az egyes típusok ábrázolása különböző architektúrákon általában különböző[1], tehát a megjelenítési réteg szolgálatai nélkül ezek a gépek nem értenék meg egymást. A

[1] lebegőpontos ábrázolás, egyes-, illetve kettes komplemens kóddal ábrázolt egészek stb.

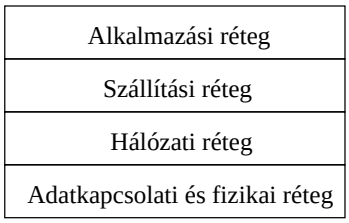
szabványos adatábrázoláson túl az adattömörítés és a kriptográfia is a megjelenítési réteg feladata.

3.2.7. Alkalmazási réteg

Az alkalmazási réteg (application layer) nagyon sok protokollt tartalmazhat. Leggyakoribb rétegfeladat például a terminálszolgáltatás, a fájlátvitel (állománytovábbítás), az elektronikus levelezés és a böngészés. A feladat a kompatibilitás biztosítása, azaz a rendszerek különbözőségéből adódó problémák leküzdése. Ezen réteg feladata megoldani a különböző rendszerek közötti fájlátvitelt. Szinte minden operációs rendszeren különböznek a fájlelnevezési konvenciók, más formában jelzik például szöveges állományokban a sorvéget, és rengeteg féle egymással inkompatibilis terminál létezik. A felsorolt és sok más nem említett probléma áthidalása mind az alkalmazási réteg feladata.

3.2.8. Adatátvitel az OSI modellben

Az OSI modell szerint, ha egy folyamat adatot akar küldeni egy másik gépen futó folyamatnak, akkor azt fizikailag nem annak, hanem az alkalmazási rétegnek küldi. Az alkalmazási réteg ellátja a saját vezérlőinformációival az adatcsomagot (ráilleszti a fejlécét), és továbbküldi a megjelenítési rétegnek. A megjelenítési réteg az előzőekben leírtak szerint sokféle transzformációt végezhet a kapott adatokon a kompatibilitás megőrzése végett, majd továbbadja az alatta elhelyezkedő viszonyrétegnek, természetesen a saját vezérlőinformációival kiegészítve. Ez a folyamat így folytatódik tovább, tehát minden egyes réteg a felette lévő rétegtől kapott adatcsomag elejére illeszti a saját vezérlőinformációit tartalmazó fejlécet, és továbbítja a neki szolgálatot teljesítő, alatta elhelyezkedő rétegnek mindaddig, amíg a már többszörösen fejlécezett csomag elér a fizikai réteghez. A fizikai réteg feladata, hogy kibocsássa a megkapott bitsorozatot a kommunikációs csatornára. A céloldalon a csomagtovábbítás hasonlóképpen történik, csak éppen visszafelé, azaz minden egyes réteg leveszi a neki szóló fejlécet az adat elől, értelmezi azokat, azok szerint jár el, majd átadja a már fejléc nélküli csomagot a felette levő rétegnek. Minden egyes réteg ezt teszi, míg el nem ér a csomag az alkalmazási rétegig. Mint kiderült tehát az adat útja gyakorlatilag vertikális, azaz a küldő gép réteges szerkezetén lefelé, a vevő állomásén pedig felfelé vezet. A rétegek azonban úgy végzik az információtovábbítást, illetve a vezérlést, mintha egyenesen a másik oldal azonos rétegének küldenék az információt. Az adattovábbítás útja tehát logikailag horizontális. Az, hogy az egyes rétegek mégis az alattuk elhelyezkedő rétegnek továbbítják fizikailag a csomagot, csak apró technikai kérdés. Az egész réteges elmélet lényege az, hogy minden réteg a vele azonos szinten levő réteggel kommunikál a protokollok felhasználásával, akkor is, ha az adatáramlás útja fizikailag függőleges irányú.



3.2 ábra A TCP/IP protokollrendszer felépítése

3.3. A TCP/IP protokollrendszer

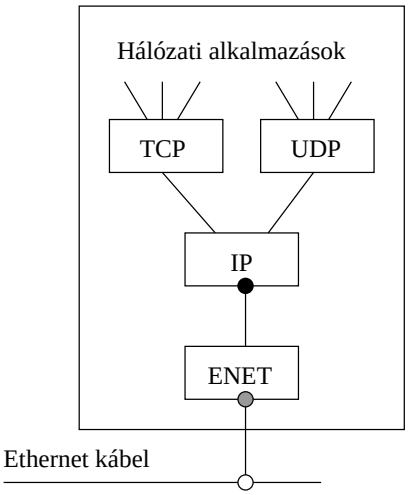
A TCP/IP protokollrendszer egy igazi, létező és működő hálózati architektúra a rétegek feladatainak, a rétegek közötti interfészek és a protokollok pontos specifikációival. A TCP/IP protokollkészlet lehetővé teszi, hogy különböző képességű, különböző gyártók által készített, teljesen különböző operációs rendszert futtató számítógépek egymással kommunikáljanak [14]. Képességei jócskán felülmúlták a vele kapcsolatos becsléseket. A fejlesztése az Amerikai Egyesült Államokban egy államilag finanszírozott, a csomagkapcsolt hálózatokkal foglalkozó kutatási projekt keretein belül indult. A kutatás során nem is remélt népszerűségre és sikerre tett szert, így válhatott egy igen széles körben elterjedt számítógépes hálózattá. A protokollrendszer specifikációja, valamint egyes implementációk ingyenesen hozzáférhetők, tehát a szó szoros értelmében nyílt rendszerről van szó.

A hálózati protokollokat általában rétegekbe szervezik, ahol minden egyes rétegnek megvan a kommunikációban betöltött szerepe [15]. Az OSI modellhez képest a TCP/IP kevesebb, alapvetően négy rétegből áll, ahogy az a 3.2 ábrán is látható. A TCP/IP-t sokkal hamarabb fejlesztették ki, mint az OSI modellt, és a tervezéskor más szempontok voltak mérvadóak. Minden egyes rétegnek megvan a saját felelőssége, amelyek lényegében hasonlóak az OSI modell azonos nevű rétegeinek a feladataihoz.

1. Az adatkapcsolati és fizikai, rövidebb de kevésbé kifejező nevén a kapcsolati (link) réteg tulajdonképpen az eszközmeghajtó (driver), amely a számítógép hálózati kártyáját az operációs rendszerbe illeszti.

2. A hálózati, vagy internet réteg a csomagok hálózaton történő mozgatását kezeli. Az útvonal-választás is ebben a rétegben van definiálva. A TCP/IP protokollrendszerben a hálózati réteget az IP (Internet Protocol), az ICMP (Internet Control Message Protocol) és az IGMP (Internet Group Management Protocol) jelenti.

3. A két hoszt számítógép között a szállítási réteg biztosítja az adatáramlást, a felette elhelyezkedő alkalmazási réteg számára. A TCP/IP protokollkészletben két alapvetően eltérő funkcionalitású szállítási protokollt különböztethetünk meg: a TCP-t (Transmission Control Protocol) és az UDP-t (User Datagram Protokol). A



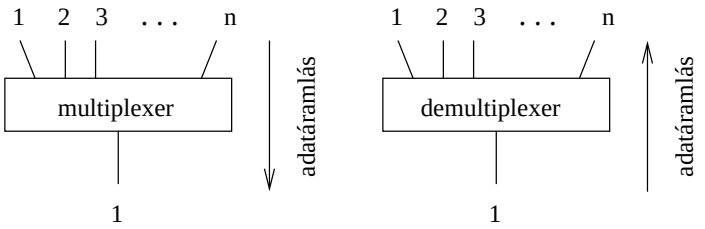
3.3 ábra TCP/IP hálózati csomópont

TCP megbízható kapcsolatot biztosít a hosztok között. A megbízhatóság eléréséhez nyugtázást, időzítéseket és számlálókat használ. Mivel ez a réteg teljesen megbízhatóvá teszi a kapcsolatot, ezzel a kérdéssel a rá épülő alkalmazási rétegnek nem kell foglalkoznia. Az UDP nem megbízható, datagram típusú szolgáltatást nyújt, amely még a csomagok helyes érkezési sorrendjét sem tudja biztosítani. Olyan alkalmazások épülnek rá, amelyeknek nem okoz gondot az esetleges sorrendcsere, a csomagvesztés, illetve a duplikálódás.

4. Az alkalmazási réteg az alkalmazások adatait kezeli. Számos olyan applikáció létezik, amely szinte minden létező implementációban megtalálható:
   - Telnet – távoli bejelentkezéshez,
   - FTP (File Transfer Protocol) – fájlátvitelhez,
   - SMTP (Simple Mail Transfer Protocol) – elektronikus levelezéshez és
   - SNMP (Simple Network Management Protocol) – hálózatmenedzsmenthez.

Az egyes rétegek a 3.3 ábrán látható módon kapcsolódnak egymáshoz egy valóságos internethosztban [16]. Az ábra egy Ethernet kapcsolaton keresztül, egy az internet-technológiát használó hálózathoz csatlakozó gép protokollrendszerének felépítését, illetve a rétegek kapcsolódását ábrázolja. Az ábrán a fehér csomópont a hálózati kártya adó-vevő egységét, a szürke az Ethernet címet, a fekete pedig az internetcímet (IP cím) jelenti. Egy-egy réteg tehát ezek alapján címezhető. Az adó-vevő egység mindent vesz, ami az Ethernet kábelen folyik, az Ethernet kártya csak a neki címzett adatokat továbbítja felfelé, az internet protokoll pedig csak a saját címére címzett üzeneteket adja át a szállítási rétegnek.

A 3.3 ábrán bemutatott protokoll-rendszerben a TCP és az UDP modulok 1–n multiplexerek, illetve n–1 demultiplexerek egyidejűleg (3.4 ábra). Multiplexerként n bemenetet

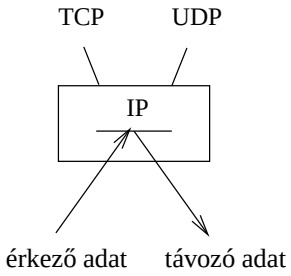


3.4 ábra 1–n multiplexer és n–1 demultiplexer

kapcsolnak egyetlen kimenetre, demultiplexerként egy bemenetet választanak szét n darab kimenetté. Ha az Ethernet meghajtóhoz (driver) a hálózaton keresztül egy Ethernet keret (frame) érkezik, akkor az továbbítódik az IP modulhoz. Az IP csomagot aszerint továbbítja a hálózati réteg a TCP-nek, illetve az UDP-nek, hogy a csomag fejlécében melyik protokoll van megjelölve. A TCP-, illetve az UDP modulhoz érkező datagram végső, alkalmazási rétegbeli célját, az alkalmazást, a datagram fejlécében található port azonosítja. Az adatút lefelé valamivel egyszerűbb, hiszen bárhonnan indul az adat, a legalsóbb rétegig egyetlen út vezet. Az UDP-, illetve a TCP modulnak azonban a lefelé vezető úton a megfelelő fejlécekbe be kell illesztenie a küldő és a címzett portok számát, amelyek alapján a túloldalon ki tudják majd választani a felfelé vezető utat, illetve tudják, hogy hova kell majd válaszolni, honnan érkezett az üzenet.

Az internetes technológia nagyon sokféle hálózati csatolót támogat, a valóságban a leggyakoribb fizikai hálózat azonban az Ethernet. Minden egyes Ethernet kártyának van egy saját, hatbájtos, az Ethernet hálózaton egyedülálló címe. Minden internet-csatlakozással rendelkező, illetve internet-technológiát használó számítógépnek van egy, az Ethernet azonosítótól teljesen független, négybájtos, az Interneten, illetve az internet-technológiájú hálózaton egyedi azonosítója. Ezek az azonosítók (címek) az adott interfészek (Ethernet, IP) alsóbb szintjén helyezkednek el. A számítógép, illetve az operációs rendszer mindig ismeri a saját Ethernet- és IP címét. Egy számítógép rendelkezhet kettő Ethernet csatolóval is. Ekkor két Ethernet-, és két IP címe van. Ebben az esetben az IP is tekinthető egy n–m multiplexernek, illetve m–n demultiplexernek. Az adat tehát érkezhet bármelyik hálózati interfész felől, és továbbítható bármelyiken keresztül. Azt a folyamatot, amikor az érkezett csomagot egy másik hálózati interfészen keresztül feldolgozás nélkül továbbítjuk, idegen nyelven *IP-forwarding*-nak nevezzük. Az IP csomagtovábbítás példája a 3.5 ábrán látható. A számítógép, amely ezt a műveletet végzi, illetve erre a feladatra van dedikálva, az internet-elv egyik legfontosabb alkotóeleme, az útválasztást megvalósító „IP-router”.

Az IP modul az internet-technológia sikerességének kulcsa. Az összes modul hozzáteszi a saját fejléc-információját a fentről érkező csomagokhoz, valamint leveszi azokat a lentről felfelé haladó csomagokról. Az IP fejléc tartalmazza az IP címet, amely révén a sok összekötött heterogén hálózatból egy egyszerű logikai hálózat képződik. A különböző



3.5 ábra IP csomagtovábbítás

hálózatok ilyen módon történő összekapcsolásáról kapta nevét a technológia: „internet”. Az egész világot behálózó, internet-technológiát alkalmazó hálózatot „Internet”-nek nevezzük.

3.3.1. Adatkapcsolati és fizikai réteg

A kapcsolati réteg feladata a TCP/IP protokoll-rendszerben, hogy IP datagramokat küldjön az IP-nek, illetve fogadjon tőle. A TCP/IP nagyon sokféle kapcsolati réteget támogat, attól függően, hogy milyen fizikai hálózat áll rendelkezésre: Ethernet, Token ring (vezérjeles gyűrű), FDDI (Fiber Distributed Data Interface), RS–232 soros vonal vagy egyéb.

A MikroWAPszerver koncepcióját tekintve a soros vonalon keresztül működő Point-to-Point Protocol (PPP), mint adatkapcsolatot biztosító protokoll, illetve réteg a legfontosabb, ezért a többi adatkapcsolati és fizikai réteg alternatívával (pl. Ethernet, Token ring) ezen dokumentum keretein belül nem foglalkozunk.

A soros porton keresztüli kommunikációt használó adatkapcsolati protokolloknak lényegében két példánya létezik: a SLIP (Serial Line IP – soros vonali IP) és a PPP (Point-to-Point Protocol – pont-pont protokoll). Időrendileg a SLIP volt az első, a PPP csak annak hiányosságait és hibáit hivatott pótolni, illetve javítani.

A PPP három fő komponensből áll:

1. az IP csomagok enkapszulációját (beskatulyázását) a soros vonalon megvalósító eljárásból,

2. egy kapcsolat-kontrolláló protokollból (LCP – Link Control Protocol), amellyel létrehozható, konfigurálható és tesztelhető az adatkapcsolati összeköttetés és

3. a hálózatvezérlő protokollok egy családjából (NCP – Network Control Protocol), amelynek tagjai például az IP, OSI hálózati réteg, DecNet és AppleTalk.

A PPP az ISO HDLC (High-level Data Link Control) szabványa szerint keretezi az információt. Egy keret minden esetben egy jelző bájttal (flag) – 0x7E kezdődik és végződik. A kezdő flag után a cím található, amely mindig 0xFF, majd egy kontrol bájt következik,

amelynek értéke 0x03. Ezután a protokollt azonosító két bájt áll. A 0x0021 például az internet protokollt azonosítja. Az információ, melynek maximális hossza 1500 bájt, csak ezek után következhet. Az adat sértetlenségét az információ után következő ciklikus redundancia kóddal ellenőrzik (CRC – Cyclic Redundancy Check), amelynek hossza két bájt. Az ellenőrző kódnál vége van a keretnek, tehát a keret végét jelző flag következik.

Mivel a keret elejét és végét jelző bájt 0x7E, a PPP-nek escape-elnie kell, ezt a bájtot a kereten belül. Szinkron kapcsolat esetében ezt a hardver megoldja, aszinkron esetben azonban a 0x7D escape karakter felhasználásával előállított, kétbájtos escape szekvencia helyettesíti a kérdéses bájtot.

A PPP célja tehát, hogy lassú soros vonalon keresztül biztosítson kapcsolatot, csökkentve a járulékos – az információ szempontjából lényegtelen – bájtok számát (overhead). A Link Control Protocolt felhasználva a legtöbb implementáció a kapcsolat felépítése során megegyezik a használatos címeket és egyes vezérlőmezőket, valamint a fejléctömörítést illetően, hogy minél kevesebb többlet információt kelljen a hasznos adatokkal egyidejűleg átvinni.

Összegezve a PPP, mint a TCP/IP egyik adatkapcsolatot biztosító protokollja a következő szolgáltatásokat nyújtja:

- egyszerre több protokoll átvitele ugyanazon a soros vonalon,
- ciklikus redundancia kód (CRC) a hibafelismeréshez és javításhoz,
- a két kommunikációs végpont IP címeinek egyeztetése,
- TCP és IP fejléctömörítés, valamint
- kapcsolat-kontrollálás (LCP), az adatkapcsolati opciók egyeztetéséhez.

3.3.2. Hálózati réteg

Az internet protokoll (IP), a TCP/IP hálózati rétege, kifejezetten a bitcsomagok forrástól a célig – összekapcsolt hálózatok rendszerén keresztül – történő továbbításához feltétlenül szükséges funkciók ellátására korlátozott protokoll [17]. Nem ad biztosítékot az adatok megbízható kézbesítésére, nem tartalmaz semmiféle folyamatszabályozási funkciót, nem határoz meg továbbítási sorrendet, és nem rendelkezik számos olyan opcióval, amelyek más ilyen jellegű hálózati protokollokban megtalálhatók. Az IP tehát egy nem megbízható és nem kapcsolatorientált, datagram alapú protokoll. Nincs garancia az elküldött adatok megérkezésére, nincs nyugtázás, de az IP a lehetőségekhez képest minden tőle telhetőt megtesz *(best effort)*. Az adatok továbbításához nem épít fel kapcsolatot, minden egyes üzenet külön utat jár. Az adatokat csomagokban (datagramokban) továbbítja, nem ad lehetőséget folyam típusú adattovábbításra.

Működés

Az internet protokoll két alapvető funkciója a címzés és a fragmentálás (tördelés). Az internet modul a címet az internet fejlécben rögzíti, ez alapján juttatható el a datagram a célállomáshoz. A továbbítási út cím alapján történő kiválasztását útválasztásnak (routing) nevezzük. Az internet modul, amikor a datagram csak kis csomagméretet engedélyező hálózatokon halad keresztül, a fragmentáláshoz és az összeillesztéshez az internet fejléc erre a célra fenntartott mezőit használja fel.

Az internet protokoll minden egyes datagramot különálló egységként kezel. Nincs kapcsolat, vagy logikai (virtuális) áramkör felépítve, amely egységbe foglalná az egyes csomagokat. Ezért nevezzük az IP-t nem kapcsolatorientált, illetve csomagkapcsolt protokollnak.

Az internet protokoll négy különböző mechanizmust használ a szolgáltatások biztosításához, melyek attribútumait a fejléc egyes mezőiben rögzíti. A szolgáltatást jellemző attribútumok az internet fejlécben szolgálat típusa *(Type of Service)*, hátralevő idő *(Time to Live)*, opciók *(Options)* és fejléc ellenőrzőösszeg *(Header Checksum)* néven szerepelnek.

Működési modell

Az egyik alkalmazás által a másiknak küldött datagram továbbításának modellje a következő forgatókönyvvel illusztrálható, ha azzal a feltételezéssel élünk, hogy az átvitel során a datagramnak egyetlen átjárón kell keresztülhaladnia:

- A küldő alkalmazás előállítja a küldendő adatot, és felszólítja az internet modult, hogy továbbítsa a közölt címre – a megadott paraméterekkel – datagramként az átadott adatcsomagot.
- Az internet modul előállítja a datagram fejlécét és az adatok elé illeszti azt. Megállapítja a célállomás lokális hálózati címét. Mivel a célállomás nincs a lokális hálózaton, az ilyenkor alapértelmezés szerinti átjáró címe a meghatározott cím.
- Kiküldi a datagramot és a lokális hálózati címet (az átjáróét) a helyi hálózati interfészre.
- A hálózati illesztő előállítja a hálózati fejlécet, a kapott adatok elé illeszti, és az így kapott adatcsomagot kiküldi a helyi hálózatra.
- A datagram megérkezik az átjáróhoz. Az átjáró hálózati csatolója értelmezi a hálózati fejlécet, s mivel az ő lokális hálózati címét tartalmazza, a neki szóló fejlécet a datagram elejéről eltávolítva továbbítja az internet modul felé. Az internet modul az internetcím alapján felismeri, hogy a datagram nem neki van címezve, tehát

továbbítania kell. Megállapítja a célállomás lokális hálózati címét és felszólítja a hálózati csatolót, hogy továbbítsa a hálózatra a datagramot.

- A hálózati illesztő előállítja a hálózati fejlécet, majd azt a kapott adatok elé illesztve kiküldi az adatcsomagot a helyi hálózatra.
- A célhoszton a hálózati csatoló megkapja a csomagot. Mivel neki címezték, a neki szóló fejléc eltávolítása után továbbítja az internet modulnak.
- Az internet modul a fejléc feldolgozása során értelmezi, hogy a kapott datagram ugyanezen hoszt egyik alkalmazásának szól, így továbbítja az adatokat, a forráscímet és egyéb paramétereket az alkalmazás felé.

Funkcionális követelmények

Az internet protokoll célja, illetve feladata a datagramok továbbítása összekapcsolt hálózatokon keresztül. Ez a datagramok szükség esetén történő továbbításával valósítható meg. Az internet modul a nem neki címzett adatokat továbbítja. Azt, hogy neki szól-e az üzenet az internetcím alapján tudja eldönteni. A továbbítások során előfordulhat, hogy a datagramnak olyan hálózaton kell keresztülhaladnia, amelynek kisebb a megengedhető csomagmérete az adott datagraménál. Ilyen esetben a csomagot több, kisebb méretűre kell szétdarabolni. E két funkció (a címzés és a fragmentálás) megvalósítása a tulajdonképpeni feladata az internet modulnak.

A címzés négybájtos (32 bites) azonosítok, számok, illetve címek alapján történik. Egy ilyen, az operációs rendszer által közölt számmal azonosítja magát az iternet modul. Ha a fejlécben található azonosító (címzett mező) megegyezik a saját címmel, akkor a csomag az adott internet modulnak szól, ellenkező esetben, ha az adott hoszt egy router, továbbítania kell, ha azonban csak egy általános internethoszt, nem szabad vele foglalkoznia. Az internetcímek öt csoportba, pontosabban osztályba sorolhatók. Az egyes osztályok „a”, „b”, „c”, „d”, illetve „e” elnevezéssel bírnak. Az osztályok megkülönböztetésének alapja a felhasználás módja, illetve annak a lokális hálózatnak a mérete[2], amelyen szét kell osztani a címeket. Az „a”, „b”, és „c” osztályok címei általános felhasználásúak, megkülönböztetésük a hálózathoz kapcsolódó hosztok számán alapszik. A „d” osztályba tartozó címek multicast célokra, az „e” osztálybeli címek egyéb, előre nem definiált feladatokra vannak fenntartva, tehát itt funkció alapján történő megkülönböztetésről van szó. Az egyes osztályok címeinek felépítése nem különösebben fontos a MikroWAPszerver szempontjából, mivel jelen esetben nem egy, az Internethez kapcsolódó hosztról beszélünk, hanem az internet-technológiát használó eszközről.

A datagramok fragmentálása (tördelése) abban az esetben fordulhat elő, amikor a forráshoszt lokális hálózatán nagyobb csomagméretek vannak megengedve, mint a továbbí-

[2] a hálózathoz kapcsolódó hosztok száma

tás során érintett hálózatokon. Ilyenkor a kisebb maximális csomagméretet engedélyező hálózatot a nagyobb csomagméretűvel összekapcsoló átjárónak a továbbítás előtt el kell végeznie a fragmentálást. Az egyes részdatagramokat valamilyen módon meg kell jelölnie, hogy a célhoszt azokat megfelelő sorrend szerint össze tudja illeszteni. A szétdarabolt datagramok összeillesztésére csak a célállomás internet moduljában kerülhet sor, nem illeszthet össze részcsomagokat egy következő átjáró, még ha a hálózaton megengedett maximális csomagméret ezt lehetővé is tenné, ugyanis az internet protokoll csomagkapcsolt mivoltából következően akár minden egyes – akár fragmentált – datagram útja különbözhet, és csak a végállomás az a pont, ahol szükségszerűen találkoznak (feltéve, hogy nem veszett el egyetlen datagram sem). Az egyes fragmentált datagramok sorszámozására az internet fejléc biztosít egy mezőt. A fragmentálásra vonatkozó utasításra, illetve információra szintén fenn van tartva egy fejléc mező. Ennek segítségével utasíthatjuk például az internet modult, hogy semmiképpen ne tördelje azt.

Átjárók

Az átjárókban implementált internet protokoll azt a célt szolgálja, hogy továbbítani tudja a datagramokat az általuk összekapcsolt hálózatok között. Egy átjáró tehát szükségképpen legalább két hálózathoz tartozik, illetve legalább két hálózati interfésszel rendelkezik. Ahhoz, hogy az átjárók kommunikálni tudjanak egymással a hatékony útvonalválasztás érdekében, szükség van egy úgynevezett átjáró–átjáró protokollra (GGP – Gateway to Gateway Protocol). Az átjárókban nincs szükség magasabb szintű protokollok megvalósítására, az átjáró–átjáró protokoll pedig az internet modul része.

IP fejléc

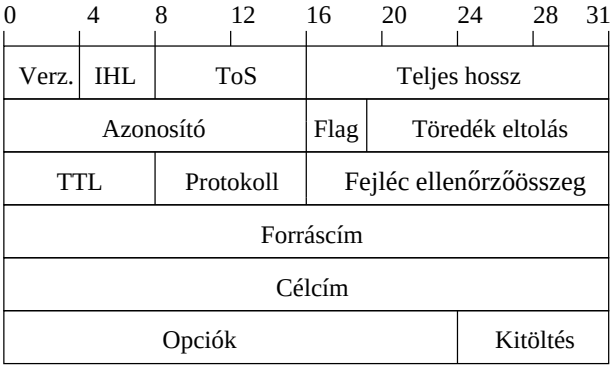
Az internet fejléc formátuma a 3.6 ábrán látható. A fejléc hossza általában 20 bájt, de különös esetben előfordulhat, hogy az opcionális „opciók” mezőt is kihasználják, ekkor a fejléc hossza növekedhet. A továbbítás bit-, illetve bájtsorrendje olyan, hogy a legnagyobb helyiértékű bit, illetve bájt a legutolsó *(big endian)*. Egy 32 bites sorozat továbbítása tehát úgy történik, hogy először a 0–7, aztán a 8–15, majd a 16–23, s végül a 24–31 számú bitek kerülnek kibocsátásra. A hálózat bájtsorrendje tehát „a legnagyobb helyiértékű bájt leghátul” típusú. Vannak olyan architektúrák, amelyek nem ebben a formában tárolják az egész számokat[3], ezért ezeknek a hosztoknak a fejlécet konvertálniuk kell a megfelelő bájtsorrendűre.

Az egyes mezők jelentése a következőkben kerül részletes kifejtésre.

Verzió – 4 bit

Az internet fejléc felépítését határozza meg. Jelenleg a 4-es változat (IPv4) az elterjedt, ezért ennek a verziónak a fejlécfelépítését tárgyaljuk.

[3] Az Intel architektúra bájtsorrendje szerint a legkisebb helyiértékű bájt van leghátul *(little endian)*.



3.6 ábra IP fejléc

IHL – 4 bit

Az internet fejléc hosszát (IHL – Internet Header Length) tartalmazza 32 bites szavakban mérve. Ha ennek a mezőnek például 5 az értéke, akkor az azt jelenti, hogy a fejléc $5 \cdot 4 = 20$ bájt hosszú, beleértve a verzió- és az IHL mezőket is.

ToS – 8 bit

A szolgálat típusa (ToS – Type of Service) mező a szolgáltatás minőségét meghatározó paramétereket tartalmazza. Arra szolgál, hogy az aktuális szolgálatparaméterek közül választani lehessen bizonyos hálózatokon keresztül történő továbbítás esetén. Számos hálózatban biztosítható a szolgáltatás precedencia. Az ilyen hálózat a magasabb precedenciával rendelkező forgalmat fontosabbnak és hamarabb továbbítandónak ítéli meg nagyobb terhelés esetén. A ToS mező három igazi választási lehetőséget kínál fel. A datagramra tehát előírható, hogy kis késleltetéssel, nagy megbízhatósággal, illetve nagy eredményességgel továbbítsa az adott szolgáltatást biztosító hálózat. Ezeket a szolgáltatásokat nemigen használják, ugyanis a legtöbb esetben a hálózat nem támogatja az előbb felsorolt szolgáltatásokat. A MikroWAPszerver szempontjából ennek a mezőnek nincs jelentősége, mert pont-pont kapcsolatban áll a két kommunikáló fél, ezért a kommunikáció során ezen mező értéke végig nulla.

Teljes hossz – 16 bit

Az internet datagram bájtokban mért hosszát jelenti, beleértve magát a fejlécet és az adatot is. Mivel 16 bit áll rendelkezésre a méret megadásához, a kezelhető legnagyobb datagram mérete 65535 bájt lehet (ez a legnagyobb, 16 biten ábrázolható szám). Az ilyen hosszú csomagok használata nem feltétlenül célszerű, azonban egy minimális méretű (576 bájt, vagy kisebb) datagramot minden hosztnak tudnia kell kezelni. Az előírt, minimális datagramméret – amelyet még tudni kell kezelni – azért 576 bájt, hogy ebben biztosan elférjen 512 bájt hasznos adat és a maximálisan 64 bájt hosszú fejléc.

Azonosító – 16 bit

A küldő fél által a datagramhoz rendelt azonosító, amelynek felhasználásával elősegíthető a fragmentált csomagok összeillesztése.

Flagek – 3 bit

Különböző vezérlőbitek. Az első bit értéke kötelezően „0”. A második és a harmadik bit a fragmentálásra utalnak. A második bit jelentése „nem fragmentálható” (DF – Don’t Fragment), a harmadik pedig azt jelzi, hogy tartozik-e még hozzá további részdatagram (MF – More Fragments). Ezen bitek „1” értéke jelenti az előbb leírtakat, „0” értéke pedig az ellenkezőjüket.

Töredék eltolás – 13 bit

Ez a mező azt jelzi, hogy a datagramon belül az aktuális részdatagram hova tartozik.

TTL – 8 bit

Hátralévő idő (TTL – Time to Live). Azt a maximális időtartamot jelzi (másodpercben), amíg az adott datagram a hálózatban létezhet. Ha a mező értéke nulla, a datagramot el kell dobni, nem szabad továbbítani, mert annak élettartama lejárt. Az internet fejléc feldolgozásakor a TTL mező értéke mindig változik. Minden egyes, a csomag fejlécét feldolgozó egységnek csökkentenie kell az értékét legalább eggyel, még akkor is, ha a feldolgozás egy másodpercnél kevesebb időt vett igénybe. A mező jelentősége abban rejlik, hogy a – valamilyen oknál fogva – nem kézbesíthető csomagok ne maradjanak meg örökre a hálózatban, maximálni lehessen az élettartamukat.

Protokoll – 8 bit

Az internet modul által kiszolgált, felsőbb protokollt azonosító szám. Minden egyes, az internet protokollt használó protokollhoz tartozik egy hozzárendelt szám [18], amely azt egyértelműen azonosítja[4]. Ez alapján dönti el az internet modul, hogy melyik felsőbb szintű protokoll felé kell továbbítania az adatot.

Fejléc ellenőrzőösszeg – 16 bit

Csakis a fejlécre számított ellenőrzőösszeg. Az abban keletkezett hibák felderítésére szolgál. Minden egyes feldolgozási művelet alkalmával újraszámítódik. A számítás során a fejléc ellenőrzőösszeg mezőjének értéke 0. Az ellenőrzőösszeg a fejléc 16 bites szavainak egyes komplemens összegének az egyes komplemense [19].

Forráscím – 32 bit

A forráshoszt internetcíme.

[4] Az UDP-hez a 0x17–es azonosító tartozik.

Célcím – 32 bit

A célhoszt internetcíme.

Opciók – változó hosszúságú

Opcionális paraméterek változó hosszúságú listája. A következő opciók állnak rendelkezésre:

- biztonsági előírások és kezelési korlátozások (katonai alkalmazások céljára),
- útválasztók utasítása az IP címük datagramba történő beírására *(record route)*,
- routerek felszólítása a feldolgozás időbélyegének beszúrására *(timestamp)*,
- kötelezően, de nem kizárólagosan érintendő útválasztók meghatározása *(loose routing)*,
- kizárólagosan érinthető routerek megszabása *(strict routing)*.

Kitöltés – változó hosszúságú

A változó hosszúságú „opciók” mezőt egészíti ki nullákkal úgy, hogy a fejléc mindenképpen 32-vel osztható darabszámú bitből álljon.

Interfészek

Minden egyes operációs rendszer különböző lehetőségeket rejt magában, ennek következtében az internet protokoll és a hierarchiailag felette álló szállítási réteg protokolljai (a TCP, illetve az UDP) közötti interfészek nem írhatók elő teljes szigorúsággal. Az interfésznek biztosítania kell, hogy el lehessen érni az internet protokoll által nyújtott összes rendelkezésre álló szolgáltatást.

A következőkben vázolt interfész eljáráshívások formájában definiálja az internet protokoll által a felsőbb protokollok számára minimálisan biztosítandó funkciókat.

- Datagram küldése
  `SEND(src, dst, prot, TOS, TTL, BufPTR, len, Id, DF, opt ⇒ result);`
  `src` – forráscím (source address)
  `dst` – célcím (destination address)
  `prot` – protokoll
  `TOS` – szolgáltatás típus (type of service)
  `TTL` – hátralevő idő (time to live)
  `BufPTR` – bufferre mutató mutató (buffer pointer)
  `len` – buffer hossza (length of buffer)
  `Id` – azonosító (Identifier)
  `DF` – nem fragmentálható (Don’t Fragment)

`opt` – opcionális adat (option data)
`result` – válasz
`OK` – datagram sikeresen elküldve
`Error` – hálózati, vagy paraméterezési hiba

- Datagram fogadása
  `RECV(BufPTR, prot`
  `⇒ result, src, dst, SpecDest, TOS, len, opt);`
  `BufPTR` – bufferre mutató mutató (buffer pointer)
  `prot` – protokoll
  `result` – válasz
  `OK` – datagram fogadása sikeres
  `Error` – paraméterezési hiba
  `len` – buffer hossza (length of buffer)
  `src` – forráscím (source address)
  `dst` – célcím (destination address)
  `TOS` – szolgáltatás típusa (type of service)
  `opt` – opcionális adat (option data)

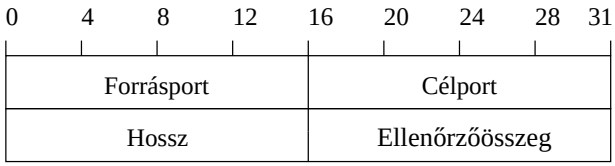
Amikor a felhasználó[5] datagramot küld, az internet modul `SEND` eljárását hívja meg az aktuális paraméterekkel. Az internet modul ezen eljáráshívás következtében leellenőrzi a hívás paramétereit és továbbítja az adatot az elkészített fejléccel. Abban az esetben, amikor a hívás paraméterei jók és a lokális hálózat elfogadja a csomagot, egy „datagram küldése sikeres” üzenettel válaszol a hívásra. Ha viszont vagy a paraméterek hibás volta, vagy a hálózat visszautasítása miatt a csomag nem továbbítható, egy „hiba” jelzéssel tér vissza.

Ha datagram érkezik az internet modulhoz (a hálózatról), és van fennálló `RECV` hívás a címzett protokoll felől, a datagramot – a fejléc ellenőrzőösszeg ellenőrzése után – továbbítja az adott szállítási protokollhoz, kielégítve a fennálló kérést. Abban az esetben, ha a címzett szállítási protokoll nem küldött a fogadásra való készenlétről tájékoztató `RECV` üzenetet, az internet modul küld egy értesítést az érkezett csomagról a címzettnek. Ha a címzett nem létezik az adott hoszton, akkor a küldő egy ICMP hibaüzenet formájában tájékoztatandó erről.

3.3.3. Szállítási réteg

A TCP/IP protokollrendszer szállítási rétegének protokolljai a TCP [20] és az UDP [21]. A TCP megbízható, kapcsolatorientált, adatfolyam típusú szolgáltatásra alkalmas

[5] A felhasználó jelen esetben a felsőbb réteg protokolljait, a TCP-t, vagy az UDP-t jelenti.



3.7 ábra UDP fejléc

protokoll. A kommunikáció kezdetén felépít egy kapcsolatot, a végén lebontja, és a kapcsolat fennállása alatt garantálja az adatok kézbesítését. A TCP-t azok a hálózati alkalmazások használják, amelyeknek a garantált adatkézbesítés elengedhetetlen. Két tipikus, a TCP-t, mint szállítási protokollt használó alkalmazás a fájlátvitel (FTP – File Transfer Protocol), és a távoli bejelentkezést biztosító TELNET. A TCP képességei a kevésbé megbízható UDP-vel szemben nagyobb processzorteljesítményt és hálózati sávszélességet igényelnek.

Az UDP nem megbízható, nem kapcsolatorientált, datagram típusú szolgáltatás ellátására alkalmas protokoll. Az alkalmazások közötti üzenetküldést minimális erőforrásigénnyel teljesíti. Nem ad garanciát a csomagok megérkezését, illetve duplikálódásának megakadályozását illetőleg. Olyan alkalmazások használhatják, amelyeknek fontosabb a gyorsaság és a kis erőforrásigény[6], mint a megbízhatóság. Tipikus, UDP-t használó alkalmazások például a hálózati fájlrendszer (NFS – Network File System), valamint a hálózatmenedzseléshez használatos SNMP (Simple Network Management Protocol).

UDP fejléc

Az UDP fejléc a multiplexitást biztosító portszámokkal, valamint az adat integritásának ellenőrzését elősegítő datagram-hosszúsággal és ellenőrzőösszeggel egészíti ki az alkalmazási rétegtől kapott adatcsomagot. A fejléc felépítését a 3.7 ábra illusztrálja. Jól látható, hogy az UDP csak minimális járulékos információt csatol az adatokhoz, ezzel is csökkentve mind a hoszt erőforrásigényét, mind a hálózati terhelést.

Port – 2·16 bit

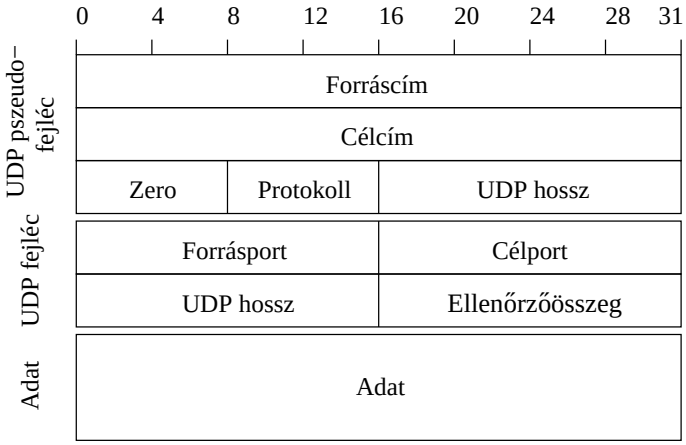
A fejléc első 32 bitje a forrás-, illetve célalkalmazások címzését lehetővé tevő portszámok által van kihasználva. Az első 16 bit a forrás-, a második a célalkalmazás portja. Mivel az UDP portszámok 16 bitesek, értékük maximálisan 65535 lehet. A 0–1023 tartomány az úgynevezett jól ismert alkalmazások számára van fenntartva [18]. Az 1023 feletti tartomány szabadon felhasználható[7]

Hossz – 16 bit

A datagram hossza bájtban mérve, beleértve a fejlécet is. A hossz mező méretéből

[6] A MikroWAPszerver viszonyrétege tipikusan ilyen.

[7] A WAP viszonyrétege például a 9200–as UDP portot használja nem kapcsolatorientált viszony esetén.



3.8 ábra Az UDP ellenőrzőösszeg számításához felhasznált adatok

következően egy UDP datagram maximális mérete 65535 bájt.

Ellenőrzőösszeg – 16 bit

Ellenőrzőösszeg, amelynek segítségével az adat integritása ellenőrizhető. A pszeudo-fejlécre, az UDP fejlécre és az adatra számított, 16 bites egyes komplemens összeg egyes komplemense. A 3.8 ábrán látható, hogy az UDP modul pontosan milyen információk felhasználásával képzi az ellenőrzőösszeget. Természetesen a számítás során az ellenőrzőösszeg mezőt *(checksum)* nulla értékkel kell figyelembe venni. UDP datagram állhat páratlan darabszámú bájtból. Ilyenkor az összeg képzése során felhasználandó 16 bites egységek közül az utolsó csonka lesz. Ebben az esetben az utolsó (páratlan) bájt egy csupa nullákból álló bájttal egészítendő ki. Az ellenőrzőösszeg értéke lehet nulla is. Ez azt jelenti, hogy a küldő UDP modulja nem számította ki az összeget, mert nem tartotta szükségesnek. Ilyen esetben természetesen szükségtelen, illetve lehetetlen leellenőrizni a datagram sértetlenségét. A szabvány szerint minden UDP modulnak ki kell tudnia számítani az ellenőrzőösszeget, a nulla értékkel történő helyettesítés csak különleges esetekben megengedett.

Interfészek

A felhasználói interfésznek lehetővé kell tennie, hogy új portot regisztrálhasson, adatot fogadhasson, illetve küldhessen a felsőbb réteg. Az adatküldésnél előírható, illetve a fogadásnál átadandó a forrás-, és a célport száma.

Az IP felé mutatott interfésznek biztosítania kell, hogy az UDP modul adatfogadáskor képes legyen meghatározni a küldő és a címzett internetcímét, valamint az internet fejléc protokoll mezőjének értékét. Egy lehetséges UDP/IP interfész az UDP fogadási kérésére

az egész internet csomagot – a fejléccel együtt – adja válaszként. Ugyanezen interfész azt is lehetővé teszi, hogy az UDP teljes, kitöltött internet fejléccel együtt küldje az adatot az internet modulnak. Az IP-nek mindösszesen ellenőriznie kell a mezőket, illetve meg kell határoznia a fejlécre az ellenőrző összeget.

3.3.4. Alkalmazási réteg

A két legrégebb óta létező, a TCP/IP protokollrendszerre épülő alkalmazás az FTP és a TELNET. Az internetes alkalmazások száma az Internet kialakulása óta folyamatosan növekszik. Nagyon sokféle szabványban rögzített alkalmazás létezik, de a TCP/IP technológiát kihasználva bárki írhat saját alkalmazást is, amely közvetlenül vagy az UDP, vagy a TCP szolgáltatásait használja ki.

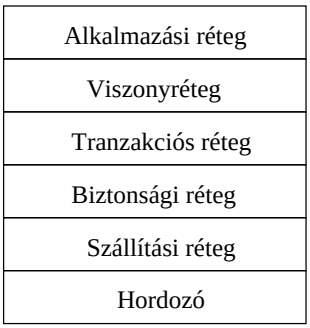
3.4. A Wireless Application Protocol

A WAP korunk két nagy ütemben fejlődő technikája, a vezetéknélküli adatelérés és az Internet konvergenciájának eredménye. A WAP adta lehetőségek kihasználásával vezetéknélküli (mobil) kliensről érhető el az Internet néhány alkalmazása. A protokollrendszere lényegében a WWW protokollrendszerét egészíti ki úgy, hogy az akár mobiltelefon számára is elérhető legyen [1].

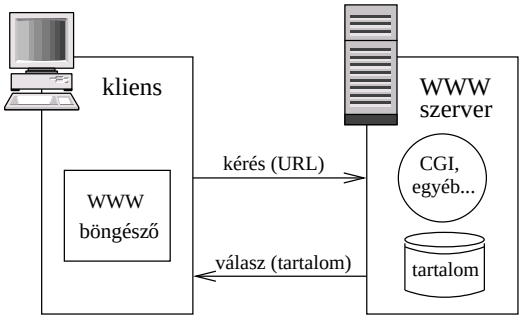
3.4.1. Felépítés

A WAP architektúrája az előzőekben ismertetett protokollokból épül fel, a protokollrendszer felépítése a 3.9 ábrán látható. Működése legkönnyebben a WWW modellel történő összehasonlítása révén ismerhető meg [22]. A 3.10 ábrán látható WWW modell szerint a kliens és a kiszolgáló között – az internet protokoll szintjén elhelyezkedő útválasztókat nem számítva, alkalmazási szinten – közvetlen kapcsolat van. A kliens a szerverhez fordul a letöltendő tartalomért, a szerver pedig a kliensnek küldi vissza válaszként a kért tartalmat. Ezzel szemben a 3.11 ábrán bemutatott WAP modell esetében a kliens és a tartalom-szolgáltató szerver között egy úgynevezett gateway, vagy más néven átjáró foglal helyet. Az átjáró feladata, hogy a kliens és a szerver protokollrendszere (nyelve) között a fordítást elvégezze. A mobiltelefon (kliens) és a gateway egymással a WAP protokollrendszerét használva kommunikál. Az átjáró és a szerver között a WWW-t kiszolgáló protokollok (HTTP[8]/TCP/IP) teremtik meg a kapcsolatot. A gateway jelenti az interfészt a két kommunikáló fél között. Felmerül a kérdés, hogy miért van szükség az átjáróra, ugyanis, ha a kliens ismerné a kiszolgáló protokollrendszerét, akkor nem kellene egy járulékos eszközt alkalmazni. A válasz kézenfekvő: egy mobiltelefon képességei egy

[8]HTTP – HyperText Transfer Protocol

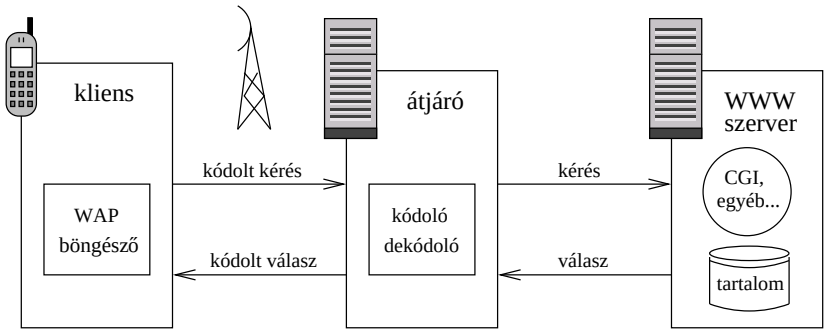


3.9 ábra A WAP rétegstruktúrája

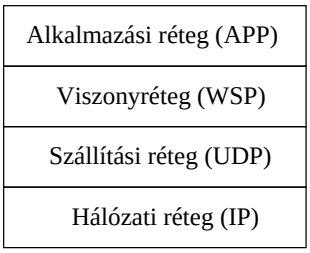


3.10 ábra WWW modell

számítógéphez viszonyítva nagymértékben korlátozottak, és az adatátviteli sebessége is sokkal kisebb. Ezért volt szükség egy új, a mobiltelefon csekély adottságait hatékonyan kihasználó protokollrendszer kifejlesztésére. A WAP vázolt protokollrendszere több alternatívát is rejt magában. A szállítási réteg protokollja lehet a TCP/IP protokollrendszer bemutatása során ismertetett UDP, illetve a WAP szabvány által specifikált WDP (Wireless Datagram Protocol [23]) is. A hordozó – WDP esetén – a rádiókommunikációban ismert számos adatátviteli eljárás (USSD – Unstructured Supplementary Service Data, SMS – Short Message Service, TETRA stb.) közül lehet az éppen rendelkezésre álló [23]. UDP esetén a hordozó természetesen csak az IP lehet. Áramkörkapcsolt adatátvitel (CSD – Circuit Switched Data) esetén a használatos protokoll profil a 3.12 ábrán látható. A mobiltelefonok körében az egyik legelterjedtebb adatátviteli eljárásról lévén szó, a MikroWAPszerver tervezésénél ezt az adatátviteli eljárást feltételeztük. A továbbiakban csak az ezen alapuló WAP protokollok ismertetésére kerül sor. A WAP viszonyrétege, a WSP (Wireless Session Protocol) engedélyezi a nem kapcsolatorientált viszonyt is, valamint a szabvány előírja ezen metódus implementálását. A WSP nem kapcsolatorientált viszony során az UDP-t használja szállítási protokollként, ezért a WTP (Wireless Transaction Protocol) – mint alternatív szállítási protokoll – sem kerül tárgyalásra.



3.11 ábra WAP modell



3.12 ábra A WAP CSD esetén használt rétegei, protokolljai

3.4.2. Wireless Session Protocol

A vezetéknélküli viszonyréteg (WSP) a szabvány [11] szerint számos szolgáltatással rendelkezik. Nem kapcsolatorientált viszony esetén ezen szolgáltatásoknak csak a töredéke kerül kihasználásra. A csak kapcsolatorientált viszony esetén kihasználható lehetőségek bemutatása ezen dokumentumnak nem célja, fontosabb az implementálandó lehetőségek ismertetése.

A WSP kifejlesztésekor az volt a cél, hogy a HTTP/1.1 [24] által nyújtott valamennyi szolgáltatás ellátására képes legyen úgy, hogy közben a mobiltelefonok architektúráját optimálisan használja ki. Az optimalizálás az átviendő és feldolgozandó adatmennyiség minimalizálásában nyilvánul meg, ezért a HTTP szöveges, tömörítetlen kommunikációjával szemben a WAP viszonyrétege bináris formában kommunikál, és amit csak lehet, tömörít.

Kapcsolat nélküli viszony szolgáltatás

A kapcsolatmentes viszony nem megerősített szolgáltatást biztosít, amely alkalmazási szinten használható fel adatcserére. Nem kapcsolatorientált viszonyban csak az eljáráshívás és a *push* szolgáltatások érhetők el. A WSP által biztosított szolgáltatás nem megerősített voltából következően a kommunikáció megbízhatósága a két fél között nem garantált.

A kliens által kezdeményezhető eljáráshívás *(method invocation)* során a hívó félnek

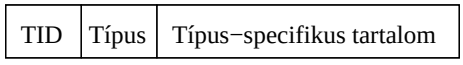
biztosítania kell a szerver és a kliens címét, a tranzakció azonosítót, a hívott metódus *(method)* kódját és a kérés URI-ját (Uniformed Resource Identifier), valamint opcionálisan megadhatók a kérés fejléc és a kérés törzs paraméterek is. A szerver címe azt az eszközt azonosítja, amelynek a kérés szól, a kliens címe pedig a kérést kezdeményező azonosítására szolgál. A tranzakció azonosító az alkalmazási réteg által használandó azonosító, amely az egyes tranzakciók megkülönböztetését teszi lehetővé. A metódus a kért műveletet azonosítja, amelynek kötelezően a HTTP/1.1 metódusai egyikének kell lennie. A kérés fejléc egy attribútum lista, amely szemantikailag megegyezik a HTTP fejléccel. A kérés törzs a kéréshez hozzárendelt adatok számára biztosít helyet.

Az eljáráshívásra adott válasz *(method result)* során megadandó paraméterek a szerver és a kliens címe, a tranzakciós azonosító, a státusz, a válasz fejléc, valamint a válasz törzs. Az eljáráshíváshoz képest új paraméter a státusz. A fejléc és a törzs ugyanazt a szerepet tölti be, csak most a válaszadó számára biztosít lehetőséget az információ közléséhez. A státusz a kérés eredményességét tükrözi. Szemantikailag azonos a HTTP által használatos státuszkóddal. A fontosabb státuszkódok és a hozzárendelt számok a függelék A.1 táblázatában találhatók meg.

A szerver által inicializálható *push* művelet során kötelező paraméterek szintén a szerver és a kliens címek, valamint a tranzakció azonosító, ami ebben az esetben a *push* azonosító nevet viseli, továbbá a *push* fejléc és törzs.

A protokoll adategység struktúrája

A WSP a WTP, illetve a WDP/UDP számára protokoll adategységekben (PDU – Protocol Data Unit) adja meg a kívánt szolgáltatáshoz szükséges összes paramétert. A PDU a 3.13 ábrán látható módon épül fel. Az első mező minden esetben a tranzakció azonosító (TID – Transaction Identifier), amely az esetlegesen fennálló több tranzakció közül azonosítja az aktuálisat, azaz megmondja, hogy a válasz melyik kérés kielégítése. Típusát tekintve egy nyolcbites, előjel nélküli egész szám. A PDU típusa az igénybeveendő szolgáltatástól függ. A fontosabb szolgáltatásokhoz tartozó típusazonosítókat a függelék A.2 táblázata tartalmazza. A típus mező szintén egy nyolcbites, előjel nélküli egész szám. A PDU további tartalma a típustól függ.



3.13 ábra WSP protokoll adategység (PDU)

Get A metódushíváshoz kétféle PDU tartozik: a *get* és a *post*. A *get* metódushívás használatos a HTTP *get, options, head, delete* és *trace* metódusának eléréséhez. A *get* PDU felépítése a 3.14 ábrán látható. Az URI hossz mező a következő mező (URI)

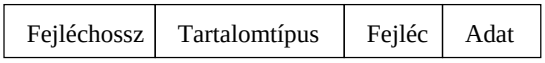
tartalmának hosszúságát adja meg bájtokban. A típusa változó hosszúságú, előjel nélküli egész[9]. A második mező az URI. A fejlécek mező a kéréshez tartozó fejléceket tartalmazza. A hossza a szállítási rétegtől kapott szolgáltatás adategység (SDU – Service Data Unit) és az előző mezők méretéből határozható meg.

3.14 ábra A WSP Get PDU felépítése

Reply A *reply* az általános válasz PDU, amelynek segítségével a kérésre adott válasz közölhető. A válasz PDU felépítése a 3.15 ábrán látható. A státusz mező a kérés eredményességét jelző egybájtos kód, amely megfeleltethető a HTTP státuszkódok valamelyikének. A fejléchossz elnevezésű mező egyértelműen a fejléc hosszúságát adja meg változó hosszúságú, előjel nélküli egész szám formájában. A tartalomtípus a válasz törzsében (adatmező) küldött tartalom típusát, formátumát határozza meg. A fejléc a válaszhoz kapcsolódó járulékos információknak ad helyet. Sikeres esetben az adatmezőben a kért tartalom, sikertelen, vagy nem tartalomra vonatkozó kérés esetén a válasz található.

3.15 ábra A WSP Reply PDU felépítése

Push A *push* és a *jóváhagyott push* PDU-k felépítése teljesen azonos. A különbség csak a PDU típus mezőjében van. A *push* PDU felépítése és a mezők célja ugyanaz, mint az előbb ismertetett *reply* PDU-nál, azzal a különbséggel, hogy itt nincs státusz mező, ugyanis nincs értelme a kérést minősíteni, hiszen nem is volt kérés.



3.16 ábra A WSP Push PDU felépítése

[9]Minden bájt első bitje mondja meg, hogy folytatódik-e a szám tovább (1), vagy az az utolsó (0). A szám a fennmaradó hétbites egységekben van eltárolva.

4. fejezet

A MikroWAPszerver fejlesztése

4.1. Analízis

Az előző fejezetben bemutatásra került a számítógépes hálózatok elméleti alapját jelentő referenciamodellt: az OSI-t, egy a valóságban széles körben használt hálózati architektúrát: a TCP/IP-t és a mobiltelefonok számára az Internetelérését lehetővé tevő szabványt: a WAP-ot. Az OSI bemutatása révén az egyes részegységek, rétegek feladata került megvilágítás alá, a TCP/IP és a WAP ismertetése pedig azért volt szükséges, mert a MikroWAPszerver ezek komponenseit kell, hogy tartalmazza a megfelelő működéshez. A MikroWAPszervert a kliens áramkörkapcsolt adathíváson (CSD) keresztül érheti el. A CSD-n keresztül történő elérés esetére a WAP specifikációja egyértelműen azonosítja a kommunikáció során használt protokollokat, miszerint a hálózati szinttől kezdve az IP, UDP, WSP és alkalmazási rétegek vesznek részt a kommunikációban. A MikroWAPszervernek ezeket a protokollokat, rétegeket illetve az azok közötti kommunikációt kell tudnia megvalósítani, mégpedig a következők figyelembevételével:

- Az internet modulnak (IP) nem kell képesnek lennie útválasztási feladatok ellátására, csupán az érkezett datagramokról kell eldöntenie, hogy neki szólnak-e. Feladata azonban a fejléc integritásának ellenőrzése, és datagram küldése esetén a másik fél számára az integritás ellenőrizhetőségének biztosítása, a fejléc ellenőrzőösszeg kiszámítása és a fejlécbe történő beillesztése révén. A szabvány szerinti internet protokoll egyik fontos funkcióját, a fragmentálást (tördelés és összeillesztés) szintén nem szükséges megvalósítania, ugyanis a WSP specifikációja szerint az átvihető legnagyobb adategység mérete 1440 bájt, és az IP 65535 bájt nagyságú datagramot is képes feldolgozni, a kliens és a szerver között pedig nincs olyan felépítésű hálózat, amely hálózati szinten nem teszi lehetővé ekkora méretű csomagok továbbítását.

- A MikroWAPszerver szállítási rétegének (UDP) a szabványban rögzített teljes funkcionalitást realizálnia kell, azaz biztosítania kell az adatok integritását (sérült adato-

kat nem továbbíthat), valamint természetesen lehetővé kell tennie az alkalmazások azonosítására szolgáló portcímzést.

- A MikroWAPszervertől elvárt funkcionalitás teljesítése érdekében elegendő, ha a WSP a szabványban rögzítetteknek csak a kapcsolatmentes viszonyra vonatkozó részeit teljesíti, a kapcsolatorientált viszonyt megvalósító rész egyszerű elhagyásával. Mivel a két különböző típusú viszony más-más UDP portot használ, azok külön alkalmazásként foghatók fel, ezért csak az egyik megvalósítása – ha a kommunikálni kívánó mobiltelefon kapcsolatmentes viszonyt kezdeményez[1] – nem rontja a WSP használhatóságát.

- Az alkalmazási rétegnek a hozzá érkező kérést a kért tartalom létezése esetén annak visszaadásával, hiánya esetén pedig a hibás kérésről szóló tájékoztatással kell kiszolgálnia, biztosítva a hardvervezérlés funkció távoli elérését is. Az OSI referenciamodell szerint az adatok reprezentációjáért a megjelenítési réteg a felelős, ezért az előállított szöveges tartalmak binárissá történő átalakítást végző fordító helye a megjelenítési rétegben lenne. Mivel sem a TCP/IP, sem a WAP nem definiál megjelenítési réteget, illetve egy újabb, egyetlen feladatot ellátó réteg bevezetésével nőne a rétegek közötti kommunikáció overhead-je, a fordító is az alkalmazási réteg részeként implementálandó.

4.2. Architekturális tervezés

A szoftver a specifikáció alapján készül három további lépésben (architekturális-, részletes tervezés és kódolás). Az architekturális tervezés annyiban különbözik a részletes tervezéstől, hogy magasabb absztrakciós szinten alkot modellt a rendszerről.

A MikroWAPszervert az előző fejezetben ismertetett réteges felépítésű hálózati architektúrák elve alapján célszerű elkészíteni. A megvalósítás során fontos az egyes rétegek feladatainak elkülönítése – nemcsak a feladatdefiníció, hanem – az implementáció szintjén is. Az egyes rétegek az előbbiek figyelembevételével külön modulokban kerülnek megvalósításra, tehát a modul kifejezés az implementáció értelmében ekvivalens a réteggel.

4.2.1. A megvalósítandó protokollok helye a hierarchiában

A MikroWAPszerver felépítése csak annyiban különbözik az ISO OSI referenciamodelltől, hogy nem definiál külön megjelenítési réteget, illetve a valójában oda tartozó

[1] A WSP viszony típusa a megadott (beállítható) portszámmal, azaz az alkalmazás címével határozható meg a kliensoldalon.

4.1 ábra A MikroWAPszerver rétegszerkezete, protokoll-hierarchiája

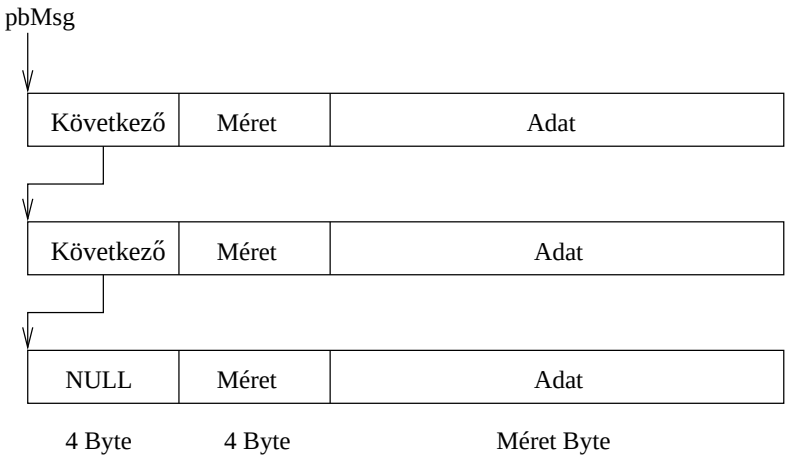
WML fordítót bizonyos architekturális kérdéseket figyelembe véve azt az alkalmazási rétegben realizálja, így tehát protokollrendszere mindösszesen hatrétegű.

Az egyes rétegek implementációi a TCP/IP és a WAP megfelelő rétegeit, illetve protokolljait valósítják meg a szabványban előírtaknak és egyes tervezési megfontolásoknak megfelelően. A protokollrendszer – ennek a diplomatervnek a keretein belül implementált rétegek kiemelt jelölésével – a 4.1 ábrán látható. A feladat tehát az adatkapcsolati réteg felett elhelyezkedő hálózati, szállítási, viszony- és alkalmazási rétegek, illetve a rétegek közötti kommunikációt biztosító protokollok implementációja.

A rétegekhez tartozó, implementálandó protokollokat a kommunikációban résztvevő másik fél, a mobiltelefon protokollrendszere határozta meg: A mobiltelefon WAP segítségével történő internetelérésének inicializálása során először áramkörkapcsolt adathíváson (CSD – Circuit-Switched Data) keresztül csatlakozik a kiszolgálóhoz, majd a pont-pont kapcsolatot biztosító PPP (Point-to-Point Protocol) segítségével összekapcsolódnak. A CSD fizikai közegként történő használata esetén, a WAP szabvány szerint, a hálózati és szállítási rétegek a TCP/IP protokollrendszerből ismert IP és UDP. Az UDP felett helyezkedik el a WSP mint viszonyréteg és az alkalmazási réteg. A tervezés és az implementáció során kihasználható az a tény, hogy a Wireless Session Protocol specifikációja előírja a nem kapcsolatorientált viszony implementálását a mobiltelefon gyártók számára. Jogos tehát az a feltételezés, hogy az összes WAP-böngészésre alkalmas mobil terminál képes a WSP-n keresztül lebonyolítható kapcsolatmentes viszony kezelésére, ezért a program erőforrásigényét csökkentendő, csak a jóval egyszerűbb kapcsolatmentes viszony implementálása a cél.

4.2.2. A rétegek közötti kommunikáció

Az egyes rétegek párhuzamosan futó processzeknek tekinthetők. Folyamatok párhuzamos futtatásához multitaszkos operációs rendszerre, de legalább egy, a kölcsönös kizárást megvalósító ütemezőre van szükség. Az ütemező megvalósítása nem tárgya en-



4.2 ábra A rétegek közötti kommunikációt biztosító üzenet-várakozási sor

nek a diplomatervnek. Az implementáció a modulok párhuzamos futtatását megvalósító ütemező meglétének feltételezésével készül.

Az egymással azonos szinten álló rétegek úgy kommunikálhatnak egymással, hogy valójában nem egymásnak, hanem az alattuk elhelyezkedő rétegeknek adják át az üzeneteket, illetve az alsóbb réteghez érkezett üzenetek továbbítódnak a megfelelő szintig. Ehhez biztosítani kell, a réteges szerkezet egyes elemei közötti üzenetátadást.

A rétegek közötti vertikális (valóságos) kommunikációt az egész rendszer szintjén egységes üzenet-várakozási sorok *(queue)* valósítják meg. A *queue*-k felépítése a 4.2 ábrán látható. A sort láncolt lista valósítja meg, melynek elemei dinamikusan foglalt memóriaterületek. Ezek első négy bájtja a következő üzenet memóriacímét, a második pedig az üzenet hosszát tartalmazza. Ha a várakozási sorban nincs több üzenet, akkor az utolsó láncszem következő üzenetre mutató mutatója „NULL” értékű. A rétegek közötti információ-továbbítás ilyen üzenet-várakozási sorok alkalmazásával kerül megvalósításra. A továbbítás során valójában nem az üzenet, hanem az üzenet által elfoglalt memóriaterület kezdetére mutató pointer kerül átadásra, így jelentős mértékben csökkenthető a feldolgozási idő. A rétegek feldolgozási műveletei során változó méretű és tartalmú csomagok a kapott üzenet kezdőcímére mutató pointerek által megcímzett memória újrafoglalásával, és a mutató átadásával kerülnek át a következő réteghez.

Egy felsőbb réteg az alatta elhelyezkedő réteg szolgáltatásait a két réteg közötti interfészen keresztül éri el, a szolgáltatást, valamint annak paramétereit előírva. A szolgálatnyújtó rétegnek szóló vezérlőinformációk azonban nem kerülhetnek bele a protokoll adatelembe (PDU), ezért az interfészeken nem a PDU-k továbbítódnak, hanem az úgynevezett interfész adatelemek (IDU – Interface Data Unit). Az IDU két részből áll: az interfész vezérlőinformációból (ICI – Interface Control Information) és az adatból. A rétegek közötti kommunikáció, azaz a szolgálatnyújtás és a szolgálatelérés tehát interfész adategységeken keresztül valósul meg.

4.3. Részletes tervezés

Az architekturális terv részleteit, pontosítását a részletes tervezés határozza meg. Az egyes rétegek között valójában megvalósuló kommunikáció, az interfészek, valamint a rétegek pontos specifikációjának meghatározása a feladata ennek a fejlesztési fázisnak.

4.3.1. PPP/IP interfész

Az internet modul a PPP-től, illetve az UDP-től kaphat üzeneteket. A PPP az előbb ismertetett üzenet-várakozási soron keresztül egyéb fejléctől és kiegészítő információtól mentes csomagokat küld. Az üzenet tehát egy internet fejlécből és a szállított hasznos adatból áll. A fejléc megegyezik a TCP/IP protokollrendszerének ismertetésekor bemutatottal, az adat pedig a fejlécben meghatározott hosszúságú tetszőleges információ. Az IP ezzel teljesen azonos felépítésű csomagokat küld a PPP-nek.

4.3.2. Internet Protocol

Ha az internet modulhoz üzenet érkezik, először megvizsgálja, hogy ki küldte. Ha a küldő a PPP, akkor a kapott csomag fejléc ellenőrzőösszegének vizsgálata után, hibátlan fejléc-információra utaló helyes összeg esetén értelmezi a címzett mezőt. Ha a csomag címzettje azonos a saját IP címével, akkor továbbítja a csomagot az UDP-nek, feltéve hogy a fejléc protokoll mezőjében az UDP kódja (0x17) szerepel. Abban az esetben, ha a fejlécben szereplő célállomás címe nem egyezik az internet modulhoz rendelt címmel, a csomagot eldobja. Valójában, hacsak valami hiba be nem következik a kliens működésében, az nem küld olyan csomagot, amelynek a címzettje nem a MikroWAPszerver, hiszen a pont-pont kapcsolat felépítése során egyeztették a címeket, a kommunikációban pedig másik eszköz nem vesz részt. A csomagot akkor is eldobja az IP, ha a protokollmezőben nem az UDP kódja szerepel, ugyanis a TCP, mint alternatív szállítási protokoll, illetve más az IP-re épülő protokoll nem szerepel a megvalósítandó architektúrában. A csomag UDP-nek történő továbbítása több lépésből áll:

- Ha az UDP előzőleg küldött az IP-nek „datagram fogadására kész” jelentésű, az IP interfészénél ismertetett *recv* üzenetet, akkor az IP automatikusan továbbítja az üzenetet az UDP felé.

- Ha azonban nincs fennálló *recv* kérés, az IP egy *notify* (értesítés) üzenet formájában értesíti az UDP-t az érkezett datagramról, a csomagot a kérés megérkezéséig egy belső üzenet-várakozási sorban tárolja, és a kérés megérkezésekor azonnal átadja az UDP-nek.

Ha az üzenet küldője az UDP, és az üzenet továbbítandó datagramot tartalmaz, akkor az internet modul ellenőrzi a megadott paramétereket, kitölti a fejlécet, kiszámítja a fejléc

ellenőrzőösszegét, és a csomagot továbbítja a PPP-nek.

Az internet modul előbbiekben ismertetett működését szemléletesen a 4.3 ábrán bemutatott folyamatábra illusztrálja.

Az internet fejléc ellenőrzőösszegének számítása

Az ellenőrzőösszeg a hálózaton keresztül kézbesített adat sértetlenségének ellenőrzését teszi lehetővé. A TCP, az UDP és az IP is ugyanazt az algoritmust használja az összeg meghatározásához.

Az ellenőrzőösszeget minden egyes kérés és válasz alkalmával ki kell számítani:

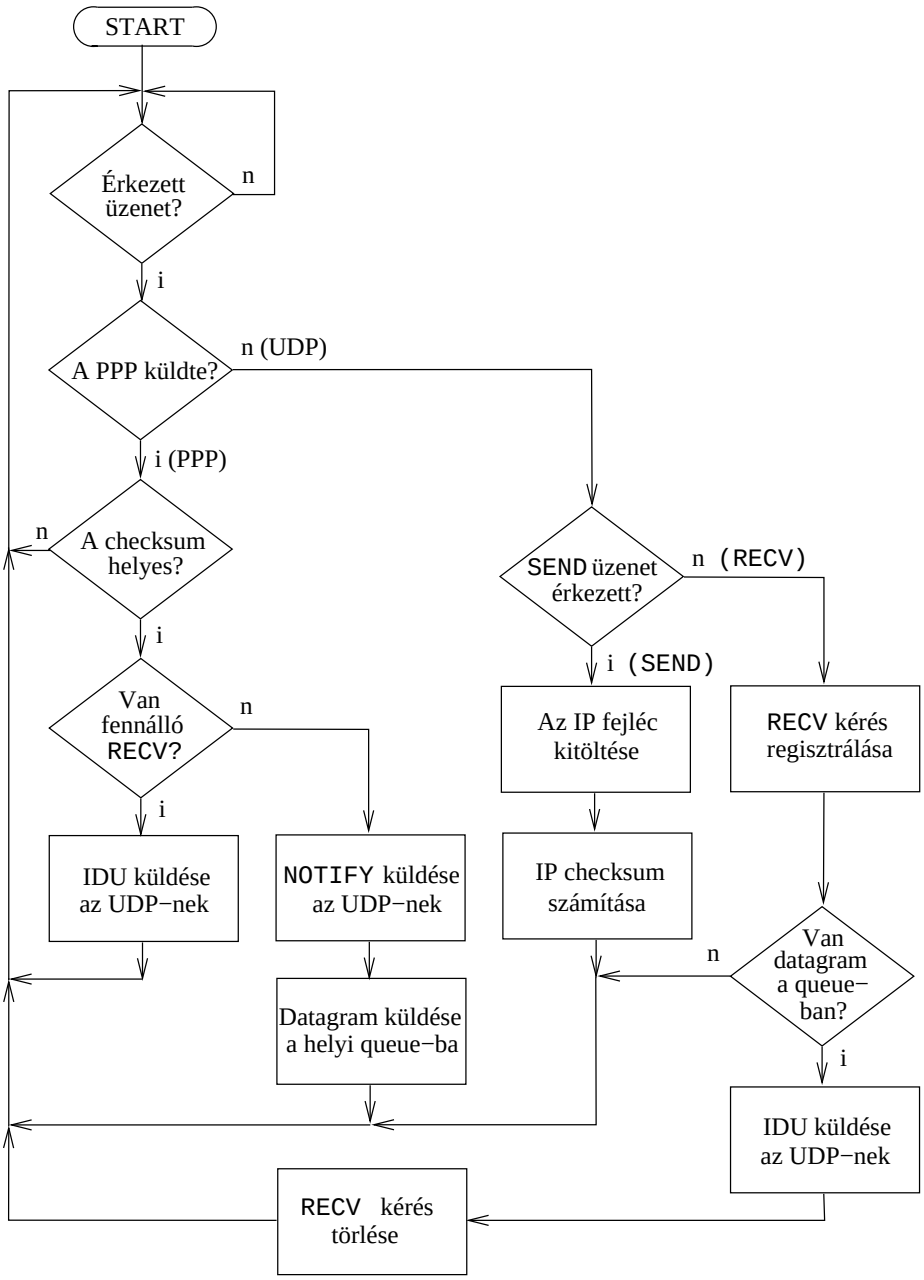
- kérés érkezésekor a kapott internet datagram fejléc integritásának ellenőrzésekor, valamint
- válasz küldésekor, hogy a másik fél szintén ellenőrizni tudja, nem sérült-e meg a datagram az átvitel folyamán.

Az ellenőrzőösszeg számítását megvalósító kód hatékonysága nagyban meghatározhatja az egész protokollrendszer hatékonyságát [19], különösen a kisebb feldolgozási képességgel rendelkező beágyazott rendszerek esetén. A fejléc ellenőrzőösszeg számítása tehát érzékeny pontja az internet protokollnak. Az összeg számításának módját egy egyszerű algoritmus írja le:

1. Az összeg kiszámítása 16 bites bájtpárokon történik egyes komplemens összegzéssel.
2. Az ellenőrzőösszeg kiszámításakor a fejléc ellenőrzőösszeg mezője üres, így az nullaként szerepel a számításban. A fejlécbe a kiszámított összeg egyes komplemense (bitenkénti inverze) kerül.
3. Az összeg ellenőrzése ugyanezeken a bájtpárokon történik – az ellenőrzőösszeg mezőt is beleértve – egyes komplemens összegzéssel. Ha az összeg minden bitje „1” (a nulla egyes komplemense), akkor az összeg helyes, illetve a fejléc hibátlan.

Az internet protokoll esetében nem fordulhat elő, hogy páratlan számú bájtra kelljen kiszámítani az ellenőrzőösszeget, ugyanis annak csak a fejlécre kell azt meghatároznia, a fejléc pedig 32 bites egységekből áll. Az UDP viszont a datagram által szállított adatot is felhasználja az összeg kiszámítása során, annak hosszúsága pedig változó lehet. Abban az esetben, ha az utolsó bájt árva, ki kell egészíteni egy nullaértékű bájttal, és ezután kell elvégezni az összeadást.

Ha az összeget az A, B, C, D, ... Y, Z jelölésű bájtokra kell kiszámítani, akkor az összegzést a (4.1) alapján kell elvégezni, ahol a $+'$ jel az egyes komplemens összeadást



4.3 ábra Az internet modul működésének folyamatábrája

jelöli. Ha az összeadandó bájtok száma páratlan, akkor a (4.2) szerint kell összegezni.

$$sum = [A, B] +' [C, D] +' ... +' [Y, Z] \tag{4.1}$$

$$sum = [A, B] +' [C, D] +' ... +' [Z, 0] \tag{4.2}$$

Kettes komplemens alapú gépen az összeadást úgy kell elvégezni, hogy a legnagyobb helyiérték felől keletkező átvitelt *(carry)* hozzá kell adni a legkisebb helyiértékű bitekhez.

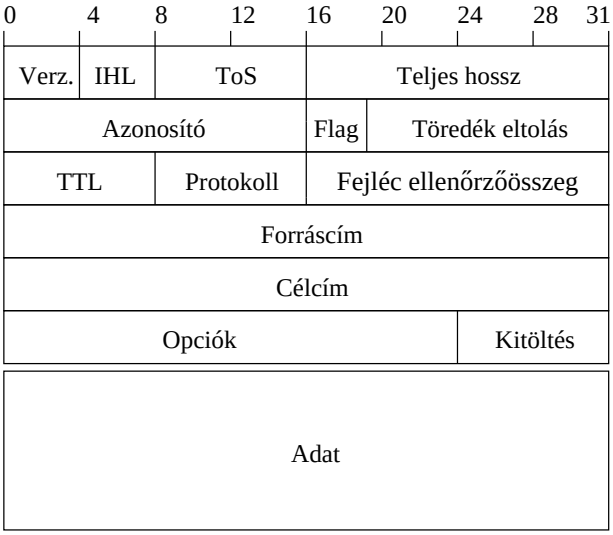
Az ellenőrzőösszeg számítása több matematikai tulajdonsággal is rendelkezik, amelyek kihasználása gyorsíthatja a művelet elvégzését:

- kommutativitás és asszociativitás – a műveletek sorrendje tetszőlegesen felcserélhető, a műveletek csoportosíthatók,
- bájtsorrend függetlenség – a művelet érzéketlen a bájtsorrendre; az összegzés után, visszarendezve a felcserélt bájtokat, ugyanaz az eredmény adódik, mint az eredeti bájtsorrenddel,
- párhuzamos összegezhetőség – olyan architektúrákon, amelyek a 16 bit többszöröse nagyságú szóhosszal rendelkeznek, az összegzés történhet $n \cdot 16$ bites egységekben is.

A 16 bites egyes komplemens összeg számítását például a következő C nyelven írt algoritmus szerint lehet elvégezni.

```
register long sum = 0;
while( count > 1 )  {
        sum += * (unsigned short *) addr++;
        count -= 2;
}
if( count > 0 )
        sum += ((unsigned short) *addr) << 8;
while (sum>>16)
    sum = (sum & 0xffff) + (sum >> 16);
checksum = ~sum;
```

A programrészlet az *addr* címtől kezdődően elhelyezkedő *count* darab, bájtra számítja ki a 16 bites egyes komplemens összeget, és a végén veszi annak az egyes komplemensét. Az összegzés 16 bites egységekben történik (a *sum* típusa long). Az első *while* ciklus összeadja a 16 bites blokkokat és az összeget egy 32 bites változóban (long) tárolja, így biztosítva a *carry* figyelembe vételét. Az *if* utasításblokkal együtt a program képes páratlan számú bájtra is kiszámítani az összeget. Ilyen esetben az utolsó bájtot egy nulla bájttal kiegészítve adja hozzá a *sum* változóhoz. A második *while* ciklus feladata az, hogy a



4.4 ábra Az IP által az UDP felé küldött üzenet formátuma

16 biten értelmezett legnagyobb helyiértéken felüli biteket *(carry)* a legkisebb helyiérték felől hozzáadja az összeghez mindaddig, amíg van további átvitel. Az ellenőrzőösszeg *(checksum)* az így kapott összeg egyes komplemense, azaz bitenkénti negáltja.

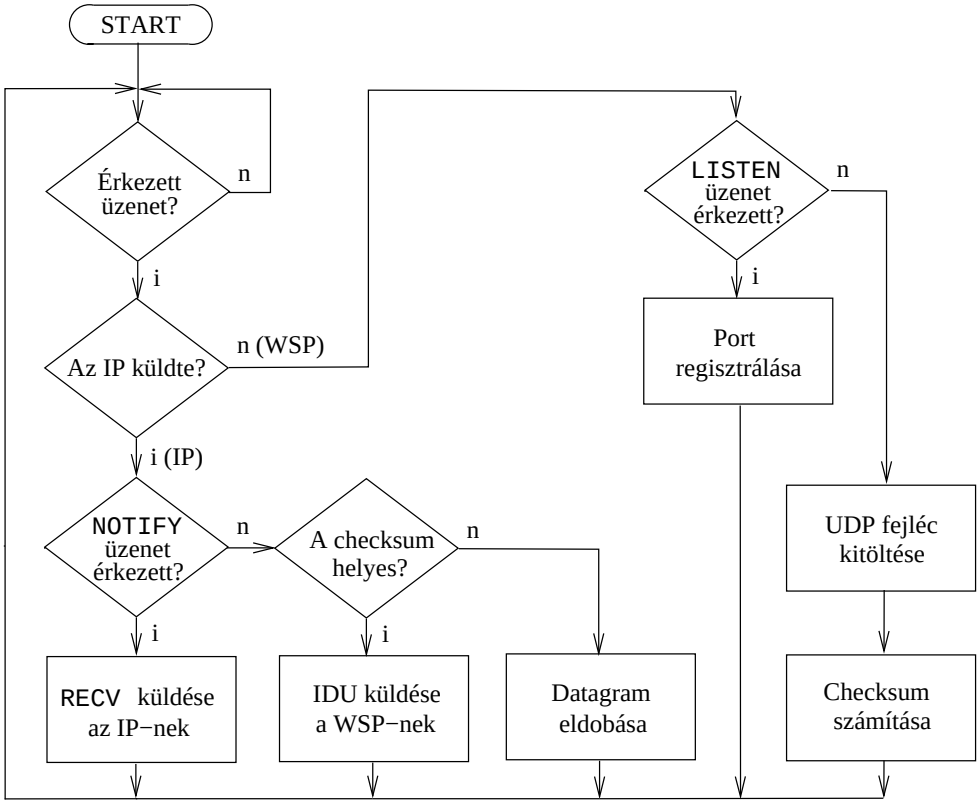
4.3.3. IP/UDP interfész

Az UDP specifikációja [21] szerint lehetséges olyan IP/UDP interfész, amely a teljes (fejléccel ellátott) IP datagram átvitelét írja elő mindkét irányban. A leírás szerint a szállítási rétegnek el kell tudni érni az internet protokoll valamennyi szolgáltatását, illetve az internet protokollnak át kell adnia minden információt, amely az IP fejlécben szerepel. A legegyszerűbb módon ez a teljes fejléc átadásával valósítható meg. A MikroWAPszerver internet moduljának működése is ezt az elvet követi, tehát az IP a 4.4 ábrán látható felépítésű csomagok formájában küldi az érkezett datagramot tovább az UDP-nek. Természetesen az UDP is ugyanilyen csomagok formájában küldi a továbbítandó adatokat.

4.3.4. User Datagram Protocol

Az UDP, az IP-hez hasonlóan, üzenet érkezésekor azt vizsgálja meg először, hogy ki küldte azt: az IP, vagy a WSP. Az IP-től érkezett csomag kétféle lehet: datagram érkezéséről tájékoztató *notify*, vagy maga a datagram. Értesítés esetén *recv* kéréssel válaszol. Ha az érkezett üzenet datagram, akkor helyes ellenőrzőösszeg esetén elküldi a WSP PDU-t tartalmazó interfész adategységet a WSP-nek feltéve, hogy a *port* fejlécmező és a WSP portszáma megegyezik. Ellenkező esetben az érkezett üzenetet eldobja.

A viszonyrétegtől kapott üzenet esetén az IDU által tartalmazott adatra, illetve a vezérlőinformációk felhasználásával kitöltött fejlécre, valamint a pszeudofejlécre kiszámítja az



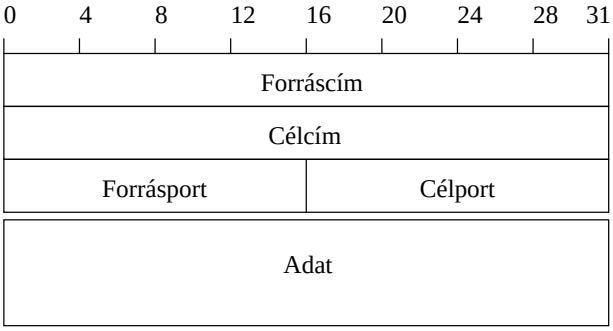
4.5 ábra Az UDP modul működésének folyamatábrája

ellenőrzőösszeget, és ha szükséges, az IP-nek szóló vezérlőinformációval kiegészíti azt, majd az így elkészült interfész adategységet adja át az internet modulnak.

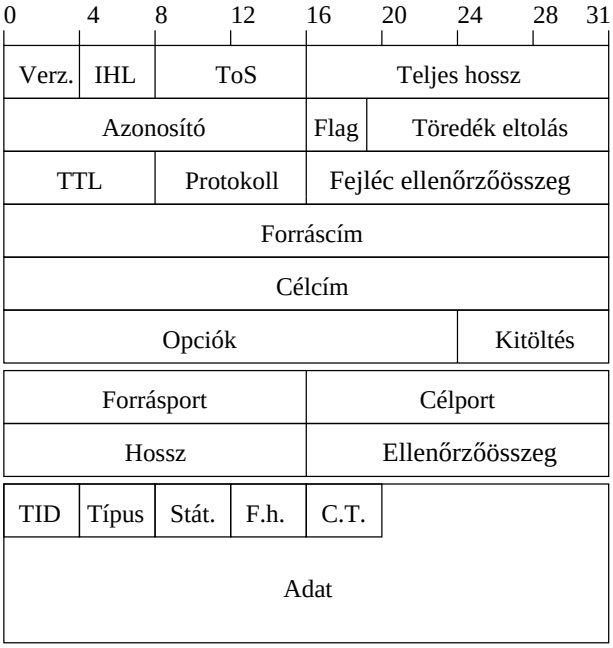
Mielőtt a WSP használni tudná az UDP szolgáltatásait, először meg kell nyitnia egy portot. A portnyitás egy *listen* üzenet segítségével történik, amely tartalmazza annak a portnak a számát, amelyet a WSP (illetve általánosan az UDP szolgáltatásait igénybe vevő protokoll) a továbbiakban használni kíván. A port azonosítása egy tetszőlegesen választott 0 és 65535 közé eső számmal lehetséges. *Listen* üzenet érkezése esetén az UDP regisztrálja a megadott számot (portot) és a továbbiakban, ha érkezik arra a portra címzett üzenet, akkor azt a megfelelő üzenet-várakozási sorba továbbítja. A User Datagram Protocol-t realizáló modul működését leíró folyamatábra a 4.5 ábrán látható.

Az UDP ellenőrzőösszeg számítása

Az UDP által előállítandó ellenőrzőösszeg számításának algoritmusa azonos az internet protokoll modul működésének ismertetésekor bemutatottal. Különbség azonban, hogy míg az internet protokoll csak a fejléc ellenőrzésére használja az összeget, az UDP az egyes datagramok által szállított adat integritását is biztosítja segítségével. Az ellenőrzőösszeg számítása során az UDP felhasználja az UDP fejlécet, az internet protokoll és az UDP egyes fejlécmezőinek tartalmát, valamint a szállított adatot.



4.6 ábra Az UDP által a WSP felé küldött üzenet formátuma



4.7 ábra A WSP által az UDP felé küldött üzenet formátuma

4.3.5. UDP/WSP interfész

Az UDP és a WSP közötti interfészre jellemző interfész-adategységek felépítését a 4.6 és a 4.7 ábrák szemléltetik. A specifikáció szerint az UDP-nek a felette elhelyezkedő réteg számára biztosítania kell a forrás- és célcímet, illetve portot datagram átadásakor. Az átadott IDU első része (az ICI) ezeket a paramétereket tartalmazza. Az IDU másik része a tulajdonképpeni WSP adategység.

Az UDP-nek biztosítania kell a saját, illetve az alsóbb rétegek összes lehetőségének elérését a felette elhelyezkedő réteg számára, ezért a 4.7 ábrán látható – az IP és UDP fejlécet is tartalmazó – üzenetformátum szerint definiálja a részére küldendő IDU-t. A WSP-nek természetesen nem kötelessége az összes mezőt kitöltenie (nem is áll rendelkezésére ehhez minden információ), az IDU felépítése azért ilyen, mert így kihasználható az összes alsóbb réteg szolgáltatása, valamint a kapott üzenetek feldolgozásakor a szolgá-

latteljesítő rétegeknek nem kell teljesen új formátumú adatcsomagot előállítania, hanem csak a már meglévő mezőket kell kitöltenie, így csökken az egyes rétegek feldolgozási ideje.

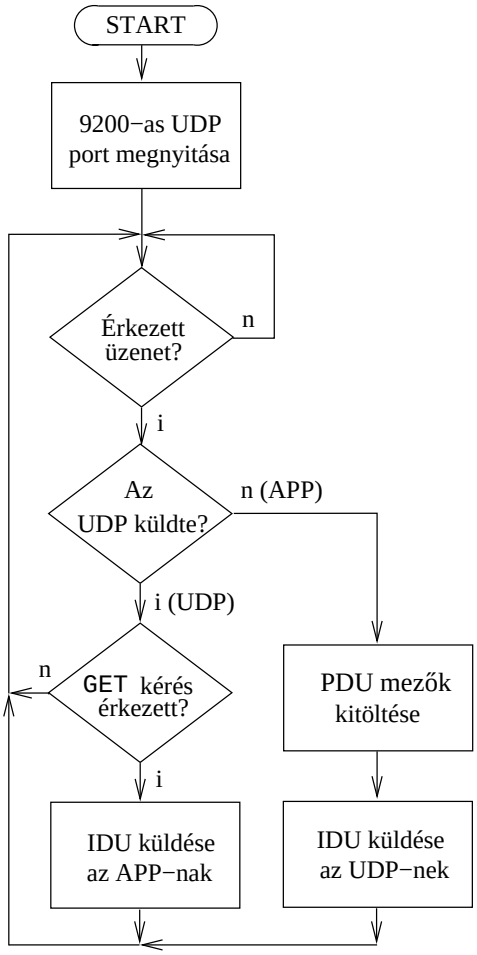
4.3.6. Wireless Session Protocol

A MikroWAPszerver részeként implementálandó WSP a szabványban rögzítettnek nagymértékben leegyszerűsített változata. A megoldás során felhasználható a WSP-nek az a tulajdonsága, amely szerint a nagybonyolultságú kapcsolatorientált viszony mellett támogatja a kapcsolatmentes viszonyt is. Ha a kommunikációban részt vevő mobiltelefonon beállítható, hogy alapértelmezés szerint kapcsolatmentes viszonyt kezdeményezzen, akkor a kapcsolatorientált viszony lehetősége mindkét oldalon kihasználatlan marad. A MikroWAPszerver alkalmazása szempontjából azonban teljes értékű kommunikáció bonyolítható le nem kapcsolatorientált viszonyban is, így a teljes WSP implementálása helyett az architektúra képességeihez, illetve erőforrásaihoz – a program, illetve a szükséges operatív memória méretét tekintve – jobban illeszkedő, kisebb tudású, csak a kapcsolatmentes viszonyt kezelni tudó WSP kerül megvalósításra, amelynek működését a 4.8 ábrán látható folyamatábra mutatja be.

A WSP csak akkor tudja a szállítási réteg szolgáltatásait használni, illetve tőle üzenetet fogadni, ha előbb regisztrálja magát a szolgálatot teljesítő protokollnál, azaz megnyit egy UDP portot. A viszonyréteget realizáló modul induláskor egy *listen* üzenettel, amelynek paramétere a port száma, megnyit egy portot. A WSP a különböző típusú viszonyokhoz (kapcsolatorientált/kapcsolatmentes) más-más portot ajánl fel hozzáférési pontként (SAP – Service Access Point). A kapcsolatmentes viszonyhoz a specifikáció ajánlása szerint a 9200-as port tartozik, tehát a portot regisztráló *listen* üzenet paramétere a 9200-as szám.

A WSP, miután regisztrálta magát a 9200-as UDP portra, képes az adatok fogadására, illetve az UDP képes a neki szóló üzeneteket továbbítani. A többi réteghez hasonlóan a WSP is először az üzenet küldőjét azonosítja. UDP-től érkező üzenet (kérés) esetén megvizsgálja a WSP adategység metódust azonosító mezőjének értékét. A mező 0x40 értéke *get* kérést jelent. A WSP nem kapcsolatorientált módban csak a *get* kérés kielégítésére képes, tehát más típusú kérést nem tud kezelni. Nem *get* metódus esetén a kéréssel a továbbiakban nem foglalkozik, nem küld hibaüzenetet sem, és a kérést tartalmazó adatcsomagot eldobja.

Az alkalmazási rétegtől csak abban az esetben érkezhet üzenet, ha azt megelőzően a WSP küldött egy kérést. A WSP definiálja a *push* metódust is, amely a kliens kérése nélkül küld tartalmat. Ilyen típusú üzenet azt megelőző kérés nélkül is érkezhet az alkalmazási rétegtől, de a MikroWAPszerver viszony-, illetve alkalmazási rétege nem tartalmazza ezt a lehetőséget. A kapott adatcsomagot a *reply* protokoll adatelem formátumára alakítása és a mezők kitöltése után továbbítja a szállítási rétegnek.

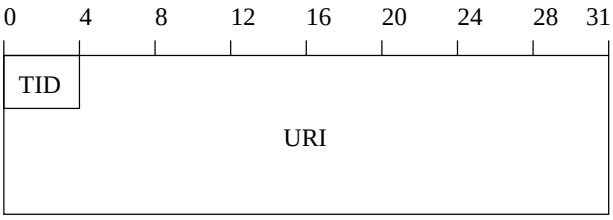


4.8 ábra A WSP modul működésének folyamatábrája

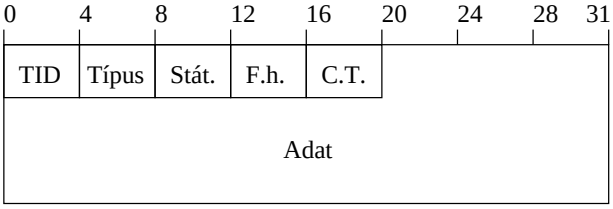
4.3.7. WSP/APP interfész

Az alkalmazási- és a viszonyréteg közötti interfészre jellemző, a két modul között kicserélődő csomagok, üzenetek felépítése a a 4.9 és a 4.10 ábrákon látható. A WSP az alkalmazási rétegnek a *get* kérés tranzakció-azonosítóját (TID – Transaction Identifier), és a kérés tárgyát képező erőforrás azonosítót (URI – Uniform Resource Identifier) küldi el. A tranzakció-azonosító átadása azért fontos, mert több, gyorsan egymás után következő kérés esetén így tudni lehet, hogy melyik kérésre adott választ az alkalmazási réteg. Ha ugyanis csak a WSP tárolná a kérést azonosítót TID-t, akkor az alkalmazási réteg esetleges hosszabb feldolgozási ideje alatt érkező második kérés azonosítója felülírná azt és a WSP az első kérés válaszát a második kérésre adott válaszként küldené vissza.

Az alkalmazási réteg által a WSP-nek küldött válasz – a tranzakció-azonosítón és a mobiltelefonon megjeleníthető dokumentumon kívül – a tartalomra vonatkozó információkat is tartalmaz. A típus mező a PDU típusát határozza meg, amely – az alkalmazási réteg szempontjából – csak *reply*, illetve *push* lehet. A PDU típusokhoz rendelt kódokat a függelék A.2 táblázata tartalmazza. A státusz mező a tranzakció sikerességéről, illetve an-



4.9 ábra A WSP által az alkalmazási réteg felé küldött üzenet formátuma



4.10 ábra Az alkalmazási réteg által a WSP felé küldött üzenet formátuma

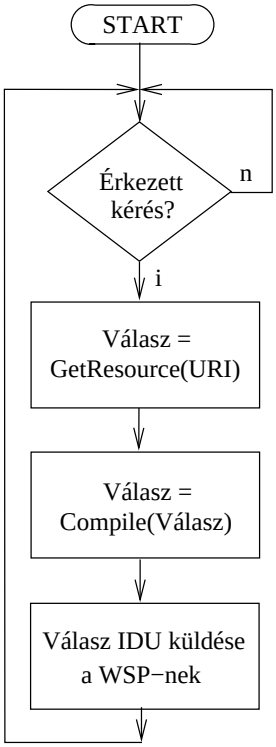
nak állapotáról ad tájékoztatást. Az egyes állapotoknak megfelelő kódok a függelék A.1 táblázatában találhatók meg. A válasz tartalmaz egy fejlécet is, amelynek hosszát a fejléc hossza mező értéke adja meg. A fejléc több információt is összefoghat, amelyek közül a legfontosabb a tartalom típusát leíró *Content-Type* mező. Minden dokumentumtípushoz különböző kód tartozik, amelyek a függelék A.3 táblázatában vannak felsorolva.

4.3.8. Alkalmazási réteg

Az alkalmazási réteg működése összetett, ezért a 4.11 ábrán látható folyamatábra csak a működés legmagasabb absztrakciós szintjét ábrázolja; az egyes részegységek működése külön kerül ismertetésre. Az alkalmazási réteg csak a viszonyrétegtől, azaz a WSP-től kaphat üzenetet. Ha kérés érkezik, akkor a tranzakció-azonosítót eltárolja, majd a *GetResource* függvény segítségével az adott URI-nak megfelelő tartalmat kikeresi, illetve dinamikus tartalomgenerálás esetén előállíttatja. A válaszként visszaküldendő tartalmat nem szöveges (WML – Wireless Markup Language), hanem bináris (WMLC – Compiled WML) formában kell prezentálni, ezért az előállított WML dokumentumot a visszaküldés előtt a *WMLCompiler* függvénnyel lefordíttatja. Az aktuális tartalomhoz kitölti a viszony- és az alkalmazási réteg közötti interfész bemutatásánál ismertetett mezőket – beleértve a tranzakció-azonosítót is –, és az előállított IDU-t elküldi a WSP-nek.

A GetResource függvény

A *GetResource* függvény feladata, hogy a paraméterként kapott URI alapján visszaadja a kért URI által meghatározott tartalmat. Ehhez két funkciót valósít meg: értelmezi az erőforrás-azonosítót, és meghívja a kért tartalmat előállító függvényt. A tartalom



4.11 ábra Az alkalmazási réteg működésének folyamatábrája

a MikroWAPszerveren függvények formájában kerül tárolásra annak érdekében, hogy könnyen lehessen elérni a WWW szervereknél ismert CGI (Common Gateway Interface) felületen keresztül futtatható programok által megvalósítható dinamikus tartalomgenerálás funkciót. A *GetResource* függvény ezeket a függvényeket hívja meg az esetleges paraméterek átadásával.

Statikus és dinamikus tartalom előállítása

A később lefordítandó, majd a mobiltelefonon megjeleníthető, még szöveges formátumú tartalom generálása történhet statikus, illetve dinamikus módon. A visszaadandó tartalmat mindkét esetben függvények generálják. Statikus esetben a tartalom előállítása teljesen egyértelmű módon, a rögzített szöveg visszaadásával, míg dinamikus esetben az átvett paraméterek függvényében történik. A tartalomgenerálást végző függvények fejléce a következőképpen néz ki:

```
void index_wml(char* Parameters, BYTE** Response).
```

A *Response* egy BYTE (unsigned char) típusú változó címét tartalmazó mutató címét tárolja. A függvény a választ nem visszatérési értékként, hanem a **Response* címtől kezdődő, dinamikusan foglalt memóriaterületen adja vissza. A *Parameters* egy olyan memóriaterület címét tartalmazza, ahol a tartalom dinamikus előállításához szükséges

paraméterek, illetve azok értékei vannak eltárolva. A paraméterek a *Parameters* karakterláncban `változó1=érték1&változó2=érték2...` formában vannak felsorolva. Ahhoz, hogy az egyes változókat, valamint azok értékeit használni lehessen, értelmezni kell a karakterlánc tartalmát. Ezt a feladatot a *GetVarFromParams*, a paramétersorból egy változó értékét karakterlánc formájában visszaadó függvény végzi el, amelynek fejléce a következő:

```
void GetVarFromParams(char* Params, char* VarNameToSearch,
                      char** VarValueToFindOut).
```

A függvény argumentumában szereplő *Params* mutató a paramétereket tartalmazó memóriaterület kezdőcímét tartalmazza. A *VarNameToSearch* változó a paraméterek közül kikeresendő változónév átadását biztosító karakterlánc kezdőcíme, a *VarValueToFindOut* pedig a megtalált érték visszaadását lehetővé tevő változó címére mutató mutató. A függvény visszatérési értéke *void*. Ha a keresett változó nem szerepel a paraméterek között, a visszaadott változó címét tartalmazó mutató értéke „NULL”. A válaszként visszaadott érték egy karakterlánc elejére mutató pointer. Ha a változó értéke számként kerül feldolgozásra, akkor azt előbb a karakterláncból át kell konvertálni, ugyanis egy szám, ha karakteres formában van tárolva, nem használható fel egyetlen, számokra vonatkozó művelet operandusaként sem. A *CastStringToInt* függvény valósítja meg a karakterláncból egésszé történő alakítást:

```
int CastStringToInt(char* String).
```

Előfordulhat azonban az is, hogy a visszaadandó, dinamikusan generált tartalomban valamilyen számmal reprezentálható értéket kell visszaadni, viszont a – később még fordítás alá kerülő – szöveges dokumentumban nem szerepelhetnek a gép számábrázolási módjától függő különböző formátumú bináris kódok. Az egészről karakterlánccá való konvertálás a *CastIntToString* függvénnyel valósítható meg:

```
void CastIntToString(int Integer, char* String).
```

A dinamikus tartalom-előállítás során keletkező kódot bufferben, azaz egy pointerrel megcímzett memóriaterületen célszerű tárolni a visszaadásig. A már részben előállított sztringhez további szöveget hozzáfűzni körülményes művelet, ugyanis az eredeti karakterlánc számára lefoglalt memóriaterületet a hozzáfűzendő karaktersorozat hosszával megnövelt nagyságúra kell újrafoglalni és az új részt az eredetihez hozzáfűzni. Ennek a műveletnek az egyszerűsítésére szolgál a *ConcateStringToBuffer* függvény:

```
void ConcateStringToBuffer(BYTE** ppbBuffer, char* pcString,
                           BYTE bSeparatorChar),
```

amely argumentumában paraméterként veszi át a memóriaterület elejére mutató pointer címét tartalmazó mutatót (*ppbBuffer*), a hozzáfűzendő karakterlánc kezdetére mutató mutatót (*pcString*) és a már meglévő és az új szövegrészt egymástól elválasztó karaktert (*bSeparatorChar*), amely lehet szóköz, újsor, vagy bármilyen tetszőleges karakter.

A bemutatott függvényekkel tehát elérhető a webkiszolgálók körében napjainkban nagy népszerűségnek örvendő technika, a dinamikus tartalomgenerálás is. A C programozási nyelv elemeinek segítségével és néhány előre elkészített függvényt felhasználva, így kihasználhatóvá váltak a CGI programok, illetve szkriptek által nyújtott lehetőségek.

Hardvervezérlés lehetősége

A tartalom dinamikus, futás közbeni előállítását függvények végzik. Ezeket a függvényeket olyan elemekkel kiegészítve, amelyek hatással vannak a mikrokontroller környezetére, megvalósítható a nagy lehetőségeket magában rejtő hardvervezérlés funkció. A MikroWAPszerver hardveréül választott mikroszámítógép képes soros portot kezelni, hiszen a mobil terminállal is ilyen interfészen keresztül kommunikál. A másik soros portjára, vagy az I/O lábaira illesztett eszköz a dinamikus tartalomgenerálást megvalósító függvények kiegészítésével vezérelhetővé, illetve lekérdezhetővé válik. A diplomatervezés során megvalósításra került egy, a hardvervezérlés lehetőségét illusztráló alkalmazás is. A kiválasztott hardver a MikroWAPszerverrel soros porton kommunikáló, mikrokontroller vezérlésű, folyadékkristályos kijelzővel, ledsorral, hangszóróval, nyomógombokkal és potenciométerekkel, rendelkező multifunkciós, programozható áramkör. Az egység működését vezérlő mikrokontroller programozásával a kijelzőn tetszőleges szöveg megjeleníthető, a ledsor bármelyik tagja külön-külön kigyújtható, a nyomógombok állása leolvasható, és a potenciométerek állapota a mikrokontrollerbe épített analóg-digitális átalakító segítségével kinyerhető. Ha a mikrokontroller programja elvégzi a soros interfészre történő illesztést, akkor az előbb felsorolt műveletek soros porton, a MikroWAPszervert a soros portjára csatlakoztatva pedig WAP felületen keresztül elérhetővé válnak. Az alkalmazás menüjét jelentő főoldal (menu.wml) a választható lehetőségeket kínálja fel. Választani lehet a kijelzőre való szövegküldés, a hangszóró repetitív megszólaltatása és az egyik potenciométer állapotának lekérdezése között. Ezeknek a funkcióknak a bemutatása illusztrálja a távoli hardvervezérlésben, illetve lekérdezésben rejlő lehetőségeket. Ha a multifunkciós próbaáramkör helyett más vezérlőegység, szabályozó, illetve beavatkozó kerül vezérlés alá, akkor bonyolult szerkezetek működése tartható kézben távolról is. Számítógépre csatlakoztatott hardver eszközök, ha a számítógép folyamatos internetkapcsolattal rendelkezik, akkor az Interneten keresztül vezérelhetők. Ha ezek a vezérelhető eszközök a háztartás részei (például fűtés, riasztó stb.), akkor az így felszerelt házat az intelligens jelzővel illetik. Az intelligens ház technikája által felkínált lehetőségeket kihasználva elképzelhető az is, hogy akár a munkahelyről hazaindulás előtt be lehessen kapcsolni a fűtést, meg lehessen nyitni a vízcsapot, és a kívánt hőmérsékletű

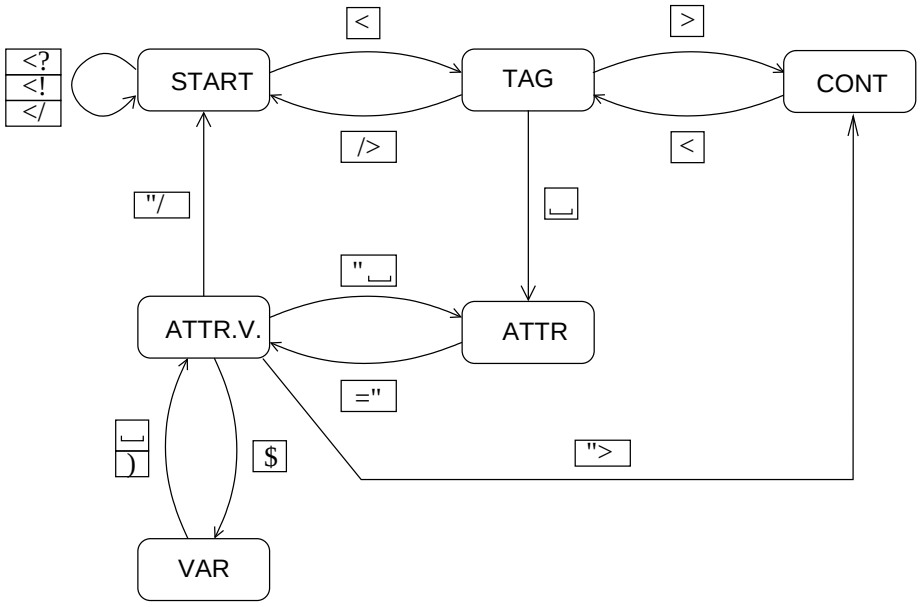
vízzel teleengedni a fürdőkádat, vagy le lehessen kérdezni a lakásban elhelyezett riasztó állapotát. A MikroWAPszerver az előzőekben bemutatott képességei alapján alkalmas az intelligens ház vezérlési feladatait ellátni úgy, hogy nem szükséges hozzá sem folyamatos internetkapcsolat, sem számítógép.

A WML fordító

A statikus, vagy dinamikus módon előállított tartalom (WML) a generálás után szöveges formátumú, olyan mint a Web esetén használt HTML (HyperText Markup Language) oldalak. A WAP lényege, hogy minimalizálja a kis sávszélességű csatornán átviendő adatmennyiséget az átviteli idő csökkentése érdekében, ezért a rendelkezésre álló sávszélességet meglehetősen pazarló módon használó szöveges formátum helyett a WAP a tényleges adatátvitelt a sokkal tömörebb bináris állományokon valósítja meg. Az átviendő bináris adat a szöveges tartalom fordítása során keletkezik. Ehhez egy olyan fordító algoritmusra van szükség, amely az egyes, a megjelenítés formázását szolgáló elemeket bináris kódokra cseréli. A fordítás azonban nem ennyire egyszerű, ugyanis egy WML dokumentum a formázó elemeken, az úgynevezett *tag*-eken kívül attribútumokat, attribútumértékeket, szöveges információkat és változókat is tartalmazhat [25]. Ezeket az elemeket, illetve ezek szekvenciáit meghatározott stratégia alapján kell lefordítani [26]. Ezt a fordítást valósítja meg az alkalmazási réteg *WMLCompiler* függvénye. A fordító általában a kliens és a kiszolgáló közötti illesztést végző átjárókban van implementálva, de a MikroWAPszerver használata esetén az átjáró funkciót is maga a MikroWAPszerver valósítja meg. Ha a kiszolgálón már előre lefordított dokumentumok (WMLC) kerülnének eltárolásra, akkor nem lenne szükség a fordítóra, és processzoridőt is lehetne spórolni vele, az alkalmazásfejlesztés dinamizmusa azonban jelentős mértékben sérülne, ugyanis már egy egyszerű statikus oldal letárolásához is egy külső fordítót kellene előbb alkalmazni, dinamikus tartalmat pedig nem lehetséges ilyen módon előállítani.

Egy WML dokumentum az előbbiek alapján tag-ekből, a tag-ek jellemzőit meghatározó attribútumokból és szöveges elemekből áll. A WML dokumentumban elhelyezett ’<’ és ’>’ jelek közötti elem a tag. Az első szó a tag neve, a többi pedig az attribútuma. A tag nem feltétlenül rendelkezik attribútumokkal, de ha igen, akkor lehet egy vagy több is belőle. Az attribútumok egy névből és az egyenlőség jel után idézőjelek közé helyezett értékből állnak. A tag-en kívüli szöveg a tartalom. Az alábbi rövid példában megfigyelhetők a felsorolt elemek.

```
<wml>
  <card id="Firstcard" title="Welcome">
    <p align="center">Hello world!</p>
  </card>
</wml>
```



4.12 ábra A WML fordító állapotgráfja

A bemutatott elemeket, illetve egységeket kell a fordítónak megkülönböztetni és lefordítani. Minden egyes tag-hez tartozik egy bináris kód a függelék B.1 táblázata szerint. Ha a tag rendelkezik attribútummal is, akkor a tag-hez rendelt bináris számhoz hozzá kell adni 0x40-et (a 6. bit „1” állapotba billentése), ha pedig tartalom tartozik hozzá, akkor a kódot további 0x80-nal kell növelni (a 7. bit értékét „1” állapotba kell állítani). Természetesen, ha tartalom és attribútum is tartozik hozzá, akkor a tag-et kódoló nyolcbites számot 0xC0-val (0x40 + 0x80) kell növelni. Az attribútumnevekhez a függelék B.2, az attribútumértékekhez pedig a B.3 táblázata szerint kell a bináris kódokat hozzárendelni. Ha a kódolandó attribútumérték nem szerepel a táblázatban, akkor egy 0x03 bájttal bevezetett és 0x00 bájttal terminált karakterláncként kell azt a bináris állományba beilleszteni.

A WML fordító a dokumentum értelmezése során különböző elemeken fut keresztül. Az egyes elemek megkülönböztetése érdekében a fordító belső állapotokkal rendelkezik, amelyek fennállása esetén különböző események hatására más-más művelet aktivizálódik. A *WMLCompiler* működését leíró állapotgép állapotait és a bekövetkező események hatására történő állapotváltásokat a 4.12 ábrán bemutatott állapotgráf szemlélteti. Az állapotátmenetek élein feltüntetett események a dokumentum olvasása során éppen beolvasott karakterpárok. Ahol eseményként egyetlen karakter szerepel az ábrán, ott a másik karakternek nincs jelentősége, az bármi lehet.

Az állapotok rövid elnevezéseinek pontos jelentései a következők:

Start Kezdőállapot. A *WMLCompiler* akkor van *Start* állapotban, amikor a fordítás kezdetén az első karaktert olvassa be, ha a tag-hez tartozó attribútum értékét nem követi a ’>’ jel után tartalom, illetve ha a tag-et követő tartalomnak vége (’<’ érkezett).

Tag A tag nevének karakterei olvasása alatt van a fordító *Tag* állapotban. A tag neve a '<' karakter után kezdődik, és az első szóköz, illetve '>' karakter beolvasása után van vége.

```
<card id="Firstcard" title="Welcome">
```

Attr *Attribútum* állapotba az állapotgép csak a tag neve (*Tag*) állapot vagy az előző tag attribútumértéke után következő szóközt követően kerülhet, tehát az attribútum nevének olvasása közben van ebben az állapotban a fordító.

```
<card id="Firstcard" title="Welcome">
```

Attrv *Attribute value* – attribútum érték. Az attribútum neve utáni egyenlőség- és idézőjelet követően kerülhet a fordító ebbe az állapotba, és addig maradhat benne, amíg az első idézőjelet el nem éri.

```
<card id="Firstcard" title="Welcome">
```

Var *Variable* – változó. Az attribútum értéke lehet változó is, illetve tartalmazhat változót. A WML-ben a változót a '$' karakter vezeti be. Az attribútum értéken belül tehát ha '$' karakter következik, akkor az állapotváltás eredménye a *Var* állapot.

```
<a href="cgi-bin/menu.cgi?menuitem=$menuitem">Click</a>
```

Cont *Content* – tartalom. Az automata a tag tartalma, azaz a megjelenítendő szöveg feldolgozása közben van a *Content* állapotban. Kezdetét a tag vége ('>'), végét a tag kezdete ('<') jelenti.

```
<p align="center">Hello World!</p>
```

Az állapotgép egyes állapotai fennállása esetén bekövetkező események hatására lezajló műveleteket a 4.1 táblázat, a *WMLCompiler* állapottáblája ismerteti.

A fordító valójában három fázisban fordítja le a dokumentumot. Az egyes fázisok külön állapotként is megjelenhettek volna az állapotgépben, de a szekvenciális működést tekintve logikusabb választás azokat az automatán kívül implementálni. A többszörös végigfutásra azért van szükség, mert az előzőekben már ismertetett módon, a fordítás során az egyes tag-ekhez különböző kódokat kell rendelni, ha azok attribútumokkal és/vagy tartalommal is rendelkeznek. Az, hogy tartozik-e egy tag-hez kiegészítő információ, csak a tag kódolása után derül ki. A fordító az első alkalommal történő olvasás során megszámolja a tag-eket, azaz az olyan '<' karaktereket, amelyek után nem '/', '?', '!' vagy újabb '<' jel következik. Erre azért van szükség, hogy tudja, mekkora tömböt kell létrehozni a tag kiegészítő elemeire vonatkozó információk tárolására. A tényleges fordítás során a tag attribútumához vagy tartalmához érve – a tag lefordítása után – az utolsó tag-hez tartozó tömbelemet növeli a megfelelő értékkel (attribútum esetén 0x80, tartalom esetén 0x40-nel), és egy másik azonos méretű tömb megfelelő indexű elemében eltárolja a tag kódjának pontos helyét a már (részben) lefordított állományban. A második átfutást követően a tag-ek attribútumainak, valamint tartalmának létezésére utaló bináris számokat

4.1 táblázat A *WMLCompiler* állapottáblája

Állapot	Esemény	Köv. áll.	Akció
Start	<?	Start	–
	<!	Start	–
	</	Start	Az általános záró tag kódjának (0x01) beszúrása.
	<	Tag	–
Tag	␣	Attr	A tag kódjának kikeresése, beszúrása; a tag helyének eltárolása. Megj.: a tag rendelkezik attribútummal.
	>	Content	A tag kódjának kikeresése, beszúrása; a tag helyének eltárolása. Megj.: a tag rendelkezik tartalommal.
	/>	Start	A tag kódjának kikeresése, beszúrása; a tag helyének eltárolása; Megj.: a tag nem rendelkezik attribútummal.
Attr	="	Attrv	Az attribútum neve kódjának kikeresése, beszúrása.
Attrv	">	Content	Az attribútum értéke kódjának kikeresése, beszúrása. Ha nincs hozzá kód rendelve, akkor karakterláncként kell beszúrni. Megj.: a tag rendelkezik tartalommal is.
	"/	Start	Az attribútum értéke kódjának kikeresése, beszúrása. Ha nincs hozzá kód rendelve, akkor karakterláncként kell beszúrni. Megj.: a tag nem rendelkezik tartalommal.
	$	Var	Az attribútum értéke kódjának kikeresése, beszúrása. Ha nincs hozzá kód rendelve, akkor karakterláncként kell beszúrni.
	"␣	Attr	Az attribútum értéke kódjának kikeresése, beszúrása. Ha nincs hozzá kód rendelve, akkor karakterláncként kell beszúrni.
Var	␣	Attrv	A változót jelző escape karakternek, és a változó nevének, mint karakterláncnak a beszúrása.
	)	Attrv	A változót jelző escape karakternek, és a változó nevének, mint karakterláncnak a beszúrása.
Content	<	Tag	A tartalom karakterláncként történő beszúrása.

tartalmazó tömb elemeinek értékeivel növeli a másik tömb azonos tömbindexű elemei által meghatározott helyen levő tag-eket reprezentáló kódokat. A *WMLCompiler* a fordítás, a kódolt állomány létrehozása során nem igényel további memóriahelyet, ugyanis a kódolt elemek a lefordítandó dokumentum elejétől kezdődően íródnak, annak helyére. A fordítás folyamatában sohasem kell visszalépni, így érdektelen, hogy a már egyszer feldolgozott területen mi van. A fordítás lényege a helytakarékosság és az átviteli sebesség növelése, ezért a WML dokumentum kódolva sokkal rövidebb; minden egyes elem kisebb, de legfeljebb az eredetivel azonos méretű helyet foglal a WMLC-ben, tehát nem fordulhat elő a még lefordítatlan tartalomra történő ráfutás sem. A fordítási műveletet követően a bináris állomány által kihasználatlanul hagyott eredeti memóriaterületet egy memória újrafoglalás művelettel lehet felszabadítani.

4.4. Kódolás

A kódolás során jön létre az előző két tervezési lépésben specifikált rendszer. Fontos a kódolás során szem előtt tartani a legfontosabb hatékonysági és biztonsági kérdéseket. Ilyenek például a legalapvetőbb programozási fogások, melyek szerint

- fontos alkalmazni az adatrejtés elvét, amely azt írja elő, hogy az adatokhoz csak az azokért felelős kód férhet hozzá,
- a kód áttekinthetősége a tesztelés során felmerülő problémák javítása, illetve esetleges továbbfejlesztés esetén kulcsfontosságú lehet, ezért a megoldást a strukturális programozás elve szerint célszerű elkészíteni,
- a specifikáció pontos betartása kulcsa az integráció sikerességének, ezért a meghatározott interfészek precíz megvalósítása nagyon fontos,
- a kódolás során nem célszerű egyes architektúrákon esetlegesen másképp, vagy egyáltalán nem működő könyvtári függvényeket alkalmazni, ugyanis ilyen esetekben sérül a kód hordozhatósága,
- függvényhívások során célszerű törekedni a cím szerinti paraméter-átadásra, ugyanis ilyenkor csak az adat memóriacímét tartalmazó pointer-t kell a stack-re helyezni, illetve onnan kiolvasni, nem pedig az esetlegesen hosszú adatot,
- dinamikus memóriahasználat során mindig ellenőrizni kell, hogy sikerült-e a szükséges memóriaterület lefoglalása,
- mivel a végső cél a program mikrokontrollerre történő átültetése, nagyon fontos, hogy a kód hatékony és kis gépi kódra lefordított programot valósítson meg.

A kódolási fázis során létrehozott modulok az integráció előtt tesztelés alá kell, hogy essenek az esetleges hibák feltérképezése érdekében. Mivel az egyes modulok (rétegek), illetve azok felületei, interfészei részletesen specifikáltak, ezért könnyen tesztelhetők. Az egyes moduloknak az interfész-specifikációban rögzítetteknek megfelelő üzenetet küldve ellenőrizhető a működés helyessége aszerint, hogy a másik interfészen ennek hatására milyen üzenetet állít elő. Hibás működés esetén a modul operációja különböző technikákkal követhető nyomon (core dump, trace, print . . .), amely művelet idegen neve: *debugging*. Fontos megjegyezni, hogy a nyomon követés nem azonos a teszteléssel. A kódolás végfázisában a modulok tesztelése és javítása a pontos, specifikáció szerinti működés eléréséig tartott.

4.5. Integrálás

Az elkészült modulok integrációja az egyes részegységek egy közös rendszerbe ágyazását jelenti. A protokollrendszer részeinek fejlesztése során az a feltételezés volt a kiindulás alapja, hogy az egyes rétegek külön processzként teljesítik feladatukat a rendszerben. Ahhoz, hogy az elkészült modulok párhuzamos taszkokként futhassanak, multitaszkos operációs rendszerre, vagy a kölcsönös kizárást és a processzek ütemezését megvalósító úgynevezett ütemezőre van szükség. Az ütemező, illetve a hálózati réteget kiszolgáló további (alsóbb) rétegek fejlesztése nem tárgya ennek a diplomatervnek, az integrációnak tehát külső modulok összekapcsolása is a feladata. Mivel az egyes részegységek standard C nyelven kerültek implementálásra, azok forrásszinten hordozhatók. Az architektúrát a hálózati és a felette elhelyezkedő rétegek implementálása során a hatékonyság és a gyorsaság szem előtt tartásán túl nem kellett figyelembe venni, hiszen a specifikáció nem tartalmaz csak az adott architektúrára alkalmazható műveleteket, és nem használ architektúrafüggő könyvtári függvényeket. Az implementált rétegek egyaránt alkalmazhatók személyi számítógépen, vagy mikrokontrolleren. Az architektúrák eltéréséből adódó inkompatibilitási problémák csak az architektúrához igazodó párhuzamos futtatást és kölcsönös kizárást megvalósító modulban, valamint a hardvervezérlés függvényeiben lehetségesek. Külön ütemező szükséges tehát a különböző architektúrákhoz, ez viszont nem csökkenti az elkészített (hálózati, szállítási, viszony-, alkalmazási) rétegek hordozhatóságát. A hardvervezérlést megvalósító CGI programokként működő függvényekben pedig a strukturális programozási technika segítségével elfedhetők a hardver különbözőségéből adódó eltérések[2]. Természetesen a végső cél a mikrokontrollerre történő átültetés volt, viszont a hordozható kód adta lehetőségeket kihasználva nem csak azon futtatható, így

[2]Különbözőképpen kell Linux vagy Windows operációs rendszer alatt megnyitni a soros portot, és másképpen kell azt kezelni mikrokontrolleres környezetben, viszont például egy *OpenSerialPort* függvénnyel a különbségek elfedhetők, mindössze az említett függvény kódja különbözik az egyes architektúrákra készített implementációkban.

tesztelése, vizsgálata könnyebbé válhat. Korlátot mindössze a mikrokontroller programtárának és operatív memóriájának nagysága és a háttértár hiánya jelent.

Minthogy az egyes részek a moduláris programozás elve szerint kerültek implementálásra, azok integrálása a modulok egyszerű beillesztésével valósítható meg. A beillesztés természetesen úgy értelmezendő, hogy az ütemező el tudja végezni feladatát a modulokon. Az interfészek és az üzenetátadás pontos specifikációjának köszönhetően az összeillesztett rendszer funkcionálisan megfelelően működött.

4.6. Tesztelés

Tesztelés alatt az integrált implementáció helyes működésének ellenőrzése értendő. A MikroWAPszerver teljesítményére vonatkozó mérések és a hatékonyságot, biztonságot tárgyaló elemzések a következő fejezetben kerülnek ismertetésre.

A szoftverteszteknek nagyon sok fajtája van, ezek is különböző csoportok szerint osztályozhatók. Egyik csoportosítási szempont a tesztelés szintje, mely szerint létezik

- modul teszt,
- alrendszer teszt,
- integrációs teszt,
- elfogadási teszt.

A modul- és integrációs tesztek a fejlesztés megfelelő (a tesztelést megelőző) fázisában hivatottak képet alkotni az implementált egységekről. A MikroWAPszerver nem tartalmaz alrendszereket, ezért ezek tesztelésére nincs szükség. Az elfogadási teszt elvégzése a mikrokiszolgáló esetében a jellegét tekintve indokolatlan.

A tesztek a szinteken túl különböző kategóriákba sorolhatók:

- funkció teszt,
- teljesítmény teszt,
- strukturális teszt.

Az előbbiek szerint a teljesítmény felmérésére vonatkozó teszteket a következő fejezet tárgyalja. A funkcionális teszt feladata ellenőrizni, hogy az elkészült implementáció működik-e egyáltalán, illetve helyesen működik-e. Mivel célul „csak” a protokollrendszer C nyelvű implementálása volt kitűzve, nem pedig a mikrokontrollerre történő átültetés, az integráció és a tesztelés is számítógépes környezetben történt. A számítógép soros portjára csatlakoztatva egy mobil terminált, az implementáció tesztelhető. A kliens

4.2 táblázat Kliens mobiltelefon beállítása

Paraméter	Érték
Telefonszám	a mobil terminál telefonszáma
Gateway IP címe	192.168.2.205
Port	9200
Felhasználónév	wap
Jelszó	wap
Adatsebesség	9600/14400bps
Vonal típusa	ISDN
Kezdőoldal	http://wap/index.wml

mobiltelefonban beállítva a kapcsolat paramétereit, az csatlakozhat a kiszolgálóhoz. A beállítások paramétereit a 4.2 táblázat tartalmazza.

Az értékekhez kiegészítésként tartozik néhány megjegyzés:

- A telefonszám annak a mobilterminálnak, vagy vezetékes modemnek a telefonszáma, amely a kiszolgáló soros portjára van csatlakoztatva. Alapértelmezés szerint egy mobil GSM terminál csatlakoztatandó a szerverhez, de a teszteléshez egy egyszerű analóg telefonmodem is alkalmazható. Így például az egy kapcsolóközponthoz tartozó végpontok közötti hívások díjtalanok, a funkcionalitás pedig tesztelhető. Tulajdonképpen a mobil terminál is csak egy modem feladatát látja el, alkalmazása a mobilitás szükségessége miatt indokolt. Ha a kapcsolat biztosításához analóg modemet alkalmazunk, akkor a vonal típusát ISDN helyett analógra kell állítani a kliens készülékben.

- A két fél a kapcsolat kialakítása során (PPP) megegyezik a kommunikáció lefolyása alatt a felek azonosítására szolgáló IP címekben. A MikroWAPszerver implementációjában a táblázatban feltüntetett érték van rögzítve, de ez természetesen tetszés szerint megváltoztatható.

- A port száma a WSP specifikációja szerint a viszony tulajdonságait jelenti. A 9200-as port használandó a kapcsolatmentes viszony létesítéséhez.

- A felhasználónév és a jelszó szintén a PPP-ben (az adatkapcsolatot biztosító rétegben) van implementálva, ezek értéke is természetesen megváltoztatható.

- Az adatsebesség GSM hálózatban a szabvány szerint 9600bps, de egyes szolgáltatók 14400bps-os sebességen (is) szolgáltatnak. Az értéket a szolgáltatóhoz alkalmazkodva kell beállítani.

- A vonal típusa mobil terminál alkalmazása esetén ISDN, analóg modemet használva pedig természetesen analóg.

- A kezdőoldal címénél valójában nincs megkötés, mert hibás kérés esetén a MikroWAPszerver a hibáról tájékoztató és a kezdőoldalra mutató linket tartalmazó oldallal tér vissza, tehát a funkcionalitás így is ellenőrizhető. Az URL (Uniform Resource Locator) hoszt része tetszőleges lehet, mert a kiszolgáló nem tesz különbséget a hoszt név, vagy cím alapján.

A kliens mobiltelefont az előbbiek szerint beállítva, és a MikroWAPszervert futtatva a számítógépen, kapcsolat létesíthető a kiszolgálóval. Megfelelően működő implementáció (helyes funkcionalitás) esetén

- a kliens és a szerver sikeresen összekapcsolódik és megegyeznek az adatkapcsolat paramétereiben,

- a kliens által elküldött kérés megérkezik a szerverhez és arra az választ generál, bármi is legyen a kérés. Hibás kérés esetén egy erről tájékoztató válasszal kell visszatérnie,

- a kialakított és válaszként visszakapott oldalakon szereplő linkek működnek, az általuk kijelölt oldalak a linkek követésével letölthetők,

- ha a hardvervezérlést megvalósító CGI programokként működő alkalmazások is a MikroWAPszerver szerves részét képezik (elvi kérdés), akkor azok működésének ellenőrzése is hozzátartozik a funkcionális teszthez. Az elkészített, egy próbapanel vezérlését végző alkalmazás tesztelése az eszközön elhelyezkedő folyadékkristályos kijelzőre történő üzenetküldéssel, a hangszóró megszólaltatásával és az egyik potenciométer állapotának a megfelelő WML oldalakon keresztül történő beolvasásával ellenőrizhető.

A MikroWAPszerver implementációja megfelel a fenti követelményeknek, a funkcionalitását vizsgáló tesztsorozat tehát sikeresnek mondható, az implementáció alkalmas egy WAP kiszolgáló feladatait elvégezni a hardvervezérlés funkció lehetővé tételével.

4.7. Összegzés

A megvalósított mikrokiszolgáló részletes tervezés után került megvalósításra. Természetesen a specifikáció apróbb hibái, illetve hiányosságai az implementáció során kerülnek felfedezésre, ezért ilyen esetben a módosítás, kiegészítés szükségszerű. A MikroWAPszerver kódolása közben is merültek fel kisebb problémák és a specifikáció annak megfelelően módosult. A részletes tervezés címszó alatt ismertetettek a végleges specifikációt tartalmazzák. A kódolás során számos szempontot kellett figyelembe venni a

hatékonyság elérése érdekében. A külön részegységként fejlesztett modulok (rétegek) összeillesztése, integrációja a pontos specifikációnak köszönhetően problémamentesen sikerült. Az elkészített mikroszerver funkcionális tesztje sikeres volt, az összes előírt követelményt maradéktalanul teljesíti, alkalmas a neki szánt feladat ellátására.

5. fejezet

Az implementáció elemzése

5.1. Hatékonyság

A nagy forgalmat bonyolító internetszerverek egyik, kívülről legfontosabb tulajdonsága a hatékonyság. A webkiszolgálók hatékonyságának, teljesítményének mérésére jól használható módszereket dolgoztak ki [27], azonban ezek az eljárások nem alkalmazhatók feltétel nélkül WAP kiszolgálók esetében, pontosan a kliensek felépítése miatt. Kliensként nem mobiltelefont, hanem annak a működését szimuláló számítógépet használva, külön a feldolgozásra és az adatátviteli sebességre vonatkozó szimulációkat is kellene alkalmazni. A MikroWAPszerver elve szerint ahhoz egyidőben csak egyetlen kliens csatlakozhat a kapcsolatot biztosító – nem multiplex működésű – GSM mobilterminál miatt. Az említett, webszerverek hatékonyságának mérésére használt módszerek, több egyszerre aktív klienssel szimulálják a nagy terhelést, és különböző mértékű terhelések esetén adódó statisztikák összevetésének eredményeként adódik az adott kiszolgálót minősítő értékelés. Mivel a MikroWAPszerver egyszerre egy kliens kiszolgálására alkalmas, a többklienses terhelés így nem alkalmazható, ezért olyan eljárásra van szükség, amely illeszkedik a MikroWAPszerver működéséhez. Léteznek a Wireless Application Protocol kiszolgálók teljesítőképességének mérésére kidolgozott módszerek is [28], azonban ezek közül is csak azok alkalmazhatók, amelyek egyszerre egy klienst szimulálva generálnak kéréseket.

5.1.1. Körülfordulási idő

A round-trip time, vagy körülfordulási idő, definíció szerint azt az időtartamot jelenti, amíg a kliensről indított kérésre megérkezik a válasz a szervertől. Ez az időtartam, ennek várható értéke, sűrűség-, illetve eloszlásfüggvénye jellemző lehet a kapcsolatra, mivel a felhasználó szempontjából a legjobban érezhető minőségi jellemző – a kiszolgálási idő – valószínűségi tulajdonságait írja le. Ezen paraméter mérése során – a MikroWAPszerver esetében is – a hatékonyságra vonatkozó következtetések vonhatók le.

A WAP protokollrendszerének felhasználásával létesített kapcsolat jellegét tekintve

speciális, ugyanis a hagyományos értelemben vett interneteléréshez képest a kliens egy képességeiben jelentős mértékben korlátozott eszköz, mind a processzorteljesítményt, az adatátviteli sebességet, a felhasználói felületet, mind a belső felépítést tekintve. A legtöbb mobiltelefonban nincs – egyes telefonokban egyáltalán – a másodpercesnél nagyobb felbontású belső óra, valamint arra sincs lehetőség, hogy az egyes bekövetkezett eseményekhez tartozó időpontokat valamilyen formában rögzíteni lehessen, ezért kliensoldalon nem lehetséges a körülfordulási idő mérése. Egy, a webszerverek körében ismert lehetőség, a CGI (Common Gateway Interface) felületen keresztül elérhető programfuttatás segítségével azonban a szerveroldali körülfordulási idő mérésére lehetőség nyílik. Szerveroldali körülfordulási idő alatt az az idő értendő, amíg a szerver válasza után azonnal generált kérés megérkezik. Ez az idő nem azonos a definíció szerinti körülfordulási idővel, de értéke megegyezik azzal, ugyanis mindkét idő a kérés elküldése és a válasz megérkezése közötti idő összege, és mivel az összeadás kommutatív, teljesen mindegy, hogy milyen sorrendben kerülnek a tagok összegzésre.

A MikroWAPszerver esetében újabb probléma vetődik fel a mérési eredmények eltárolása kérdésének megfontolásakor, ugyanis számítógépen futó kiszolgálóknál könnyedén megoldható, hogy a mérési pontok például egy fájlban rögzítésre kerüljenek, a mikrokontroller viszont nem tartalmaz olyan háttértárat, amely futás közben írható lenne, és az adatokat statikusan tárolja. A mikrokontroller kétféle memóriával rendelkezik: programtárral és operatív memóriával. A programtár a legtöbb mikrokontroller esetében futás közben nem, csak a program feltöltése során írható. Léteznek olyan mikrokontrollerek is, amelyek programtára írható futás közben is, ugyanis programból átkapcsolható memóriaírásra, ekkor viszont kikerül a futási állapotból, és a mérés során nem csak a tiszta kiszolgálási idő kerülne bele az eredménybe. Az operatív memória elég kisméretű ahhoz, hogy a futás közben szükséges változók, a verem és a dinamikusan foglalt memóriaterületek számára biztosított helyen felül ne maradjon hely nagyobb méretű statisztika készítéséhez felhasználható, nagyszámú mérési eredményt tartalmazó adatblokk tárolására, valamint további megoldandó kérdést jelentene az adatok áttöltése és feldolgozása.

A körülfordulási idő mérése legegyszerűbben úgy valósítható meg, hogy a kliensről indított kérésre a kiszolgáló olyan választ állít elő, amelynek kliensoldali feldolgozásának következtében a kliens egy újabb kérést generál a szerver felé. Az egyszer elindított folyamat csak a szerver, vagy a kliens futásának leállításával állítható le. Ha a szerver a válasz előállítása során a mikrokontroller soros portján, vagy valamelyik I/O lábán egy, például számítógép számára észlelhető jelet bocsát ki, akkor a futási időpontok a MikroWAPszerverhez csatlakoztatott számítógépen rögzíthetők. A mérés automatikus, mert az egyszer kibocsátott kérés hatására a folyamat végtelen ciklusba kerül, valamint a kliens és szerveroldali adattárolás problémája is megoldódik ezzel a módszerrel. A mérés folyamata tehát a következő:

1. A kliens kérést indít a szerver felé.

2. A szerverhez megérkezik a kérés, és ennek hatására előállítja a visszaadandó tartalmat.

3. A válasz visszaküldése előtt – nem a mobil terminál által lefoglalt, hanem – a másik soros porton kiküld egy bájtot jelzésként.

4. (a) A soros portra illesztett számítógép észleli az eseményt, és egy fájlba letárolja a jelzés pillanatának időpontját.

   (b) A szerver, anélkül hogy várna bármiféle visszajelzésre a számítógép felől, visszaküldi az előbb előállított tartalmat a kliensnek.

5. Megérkezik a válasz a klienshez, és az feldolgozza, majd megjeleníti azt.

6. A feldolgozott WML dokumentum által tartalmazott időzítő[1] lejárásához társított esemény aktivizálódik, amely arra szólítja fel a klienst, hogy ismét küldjön kérést a kiszolgálónak.

7. A kérés elküldése után a folyamat kezdődik elölről.

A kért tartalom visszaadását – az előző fejezetben ismertetett módon – függvények valósítják meg. A CGI programok funkcionalitása szintén elérhető ezen függvények felhasználásával, tehát a körülfordulási idő mérése is megoldható segítségükkel. Az előbbiekben ismertetett forgatókönyv szerveroldali részét a következő kódrészlet valósítja meg.

```
void rtt_cgi(char* Parameters, BYTE** Resp)
{
  HANDLE hCom2 = OpenSerialPort(COM2_PORT);
  WriteByteToSerialPort(hCom2, 0x07);
  CloseSerialPort(hCom2);
  ConcateStringToBuffer(Resp,
    "<wml>"
    "<card id=\"test\""
    "ontimer=\"http://microwapserver/rtt.cgi\""
    "title=\"RTT\"><timer value=\"1\"/>"
    "<p align=\"center\">Round-Trip time</p>"
    "</card>"
    "</wml>", '\0');
}
```

[1] A WML specifikációja szerint az időzítő 0.1s-os felbontással paraméterezhető, és az időzítés értéke nem lehet nulla. A későbbiekben rögzített időpontok különbségéből kivonható a késleltetés 0.1s-os értéke és így az nem hamisítja meg a mérést.

A függvény argumentuma azonos az implementációt ismertető fejezetben bemutatottal, mely szerint a paramétereket tartalmazó karakterláncot és a válasz visszaadását biztosító memóriaterület elejére mutató pointer címét veszi át. A bemutatott kód láthatóan nem elemi C utasításokat, hanem függvényhívásokat tartalmaz. Az *OpenSerialPort* függvény megnyitja az argumentumában paraméterként megadott soros portot és annak elérését biztosító úgynevezett *handler*-rel tér vissza. A *WriteByteToSerialPort* utasítás a megadott *handler*-hez tartozó soros portra egyetlen bájt kiírását teszi lehetővé. A *CloseSerialPort* értelemszerűen a port lezárását valósítja meg, a *handler* felszabadításával. A tartalom előállítását, illetve bufferbe történő tárolását a *ConcateStringToBuffer* függvény oldja meg. Az *rtt_cgi*-t meghívó függvény, annak visszatérése esetén a válasz tárolására szolgáló bufferből olvassa ki a kliensnek elküldendő tartalmat. Ezzel a kódrészlettel tehát megvalósítható az, hogy erre vonatkozó kérés érkezése esetén a MikroWAPszerver másik soros portján egy egyszerű jelzéssel értesíthető legyen az arra csatlakoztatott számítógép, megoldva így a loggolás architektúrából adódó problémáját. Azt, hogy a kliens a kapott oldal feldolgozása után ismét kérést küldjön a kiszolgálónak a visszaadott WML oldal garantálja, ugyanis az abban elhelyezett időzítő lejárásához van társítva az újabb kérés generálása. A minden egyes (egymás után gyorsan következő) kérés alkalmával történő portnyitás késleltetést vihet a rendszerbe, ezért a mérés idejére megoldható az, hogy a szerver induláskor nyissa meg a portot és csak a leállításkor zárja azt le.

A MikroWAPszerver teljesítményére és hatékonyságára vonatkozó méréseket – mivel az implementáció nem lett mikrokontrollerre átültetve – csak számítógépes környezetben lehet elvégezni. Egy számítógép nyilvánvalóan nagyobb erőforrásokkal rendelkezik, ezért valójában a mérési eredmények a mikrokontrollerre átültetendő MikroWAPszerverre nem jellemzők, csakis tájékoztató jellegűek, céljuk becslést adni arra, hogy a kliensről inicializált tartalom-letöltéshez, illetve az egyes részműveletekhez szükséges idők hogyan arányulnak egymáshoz.

A mérések során felhasznált eszközök paramétereit az 5.1 táblázat tartalmazza. Meglehetősen nagy különbség látható a szerverként és kliensként alkalmazott számítógépek teljesítménye között. A kiszolgáló szerepét betöltő számítógép lassabb processzorral és kisebb operatív memóriával rendelkezik, mert így képességeiben közelebb áll a végleges megvalósításhoz tervezett mikrokontrollerhez. A kliens számítógép nagyobb teljesítményét pedig az indokolja, hogy az ne okozzon a mérések során a kapcsolat véges sebességéből és a kiszolgáló feldolgozásából adódó késleltetéssel összemérhető nagyságú többletkésleltetést, hiszen nem a kliens, hanem a szerver hatékonyságának mérése a cél. A kapcsolatot két, a táblázatban látható típusú mobiltelefon biztosította a méréshez, így a csatorna átviteli sebessége azonos a végleges alkalmazás során felhasználttal. A fejezetben a továbbiak során ismertetett mérések alkalmával is ezen táblázat által tartalmazott jellemzőkkel rendelkező eszközök kerültek felhasználásra.

Az 5.2 táblázatban a MikroWAPszerverhez tervezett eszköz, illetve eszközök erőfor-

5.1 táblázat Méréshez felhasznált eszközök paraméterei

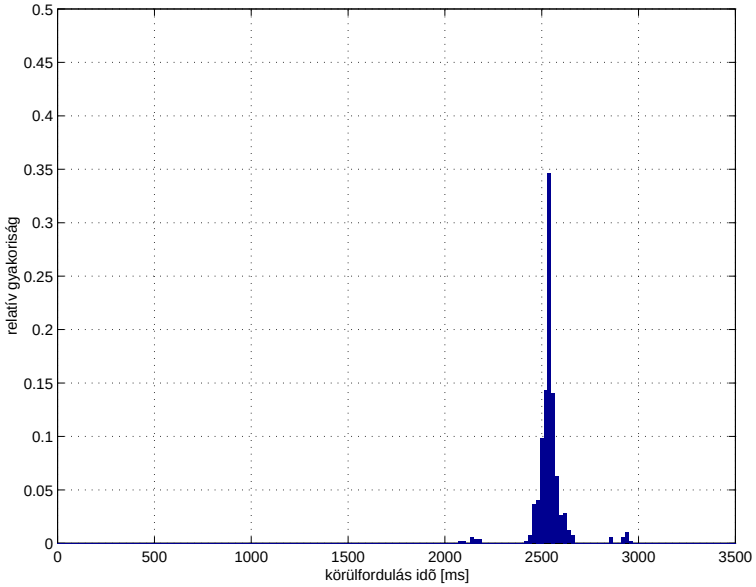
	Kliens	Szerver
Processzor	600MHz	100MHz
Memória	128MB	16MB
Szoftver	Nokia WAP Toolkit 2.1	MikroWAPszerver 1.84
Modem	Motorola Timeport P7389i	Motorola Timeport P7389
Kapcsolat típusa	ISDN	ISDN
Adatsebesség	9600bps	9600bps

5.2 táblázat A MikroWAPszerver tervezett erőforrásai

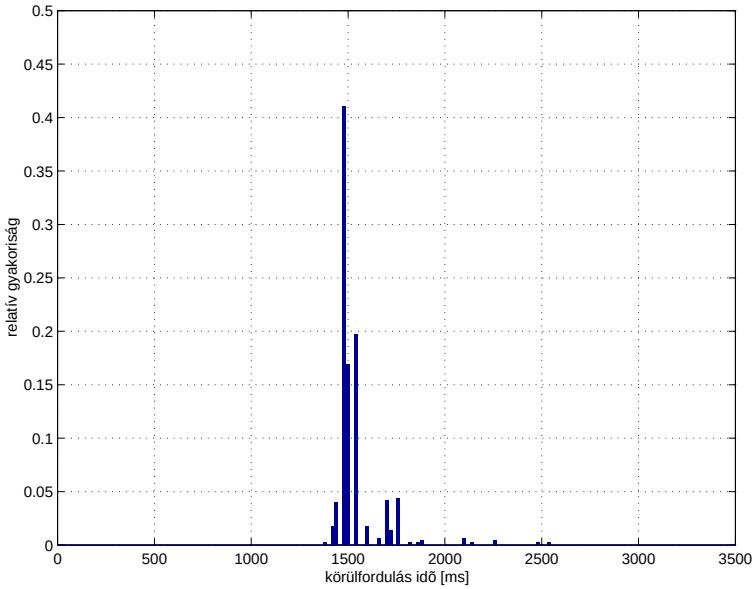
Erőforrás	Érték
Processzor	14.7MHz
Programtár	128kB
Memória	128kB
Szoftver	MikroWAPszerver 2.0
Modem	Siemens M20T

rásai vannak felsorolva. Összehasonlítva a számítógépes kiszolgáló és kliens képességeit, lényeges különbség a processzor sebességében van. A programtár mérete annyiban fontos, hogy el kell férnie benne a kiszolgálót megvalósító programnak, az operatív memória „kis” mérete pedig nem jelent korlátozást, hiszen a legnagyobb (alkalmazási szinten) kezelendő csomag mérete 1440 bájt, és ez kiegészítve az egyes rétegek információival, fejléceivel is jóval a megengedett határ alatt marad. A táblázatban feltüntetett modemként szereplő mobil terminál pedig annyiban különbözik egy hagyományos mobiltelefontól, hogy nem rendelkezik kezelőfelülettel, nincs tehát sem billentyűzete, sem kijelzője; a MikroWAPszerverhez való kapcsolódást a soros interfésze teszi lehetővé.

A körülfordulási idő relatív gyakoriságfüggvénye egy valóságos, teljes – kliensből, átjáróból és az Interneten valahol elhelyezkedő webszerverből álló – rendszerre az 5.1(a), a MikroWAPszerverre pedig az 5.1(b) ábrán látható. Az összehasonlíthatóság miatt a két rendszerre vonatkoztatott függvények ugyanazon az időtartományon vannak ábrázolva. A MikroWAPszerveren mért körülfordulási idő relatív gyakoriságfüggvényének jellege azonos a kliens-átjáró-szerver együttes kürölfordulási idő statisztikájának jellegével. Az értékekben mutatkozó, viszonylag nagynak mondható különbség annak köszönhető, hogy míg a MikroWAPszerver esetében az átjáró és a szerver fizikailag egy eszközben van, továbbá a kliens és a szerver között fix sebességű vonal létesít kapcsolatot, addig a másik esetben akár több száz km távolság is lehet a két fél között, a késleltetés pedig a távolságtól és az Internet pillanatnyi forgalmától egyszerre függ. A késleltetés az Internet jellegéből adódóan nem csak nagyobb, hanem például napi átlagot tekintve sokkal nagyobb



(a) teljes, kliens-átjáró-szerver rendszeren



(b) a MikroWAPszerveren

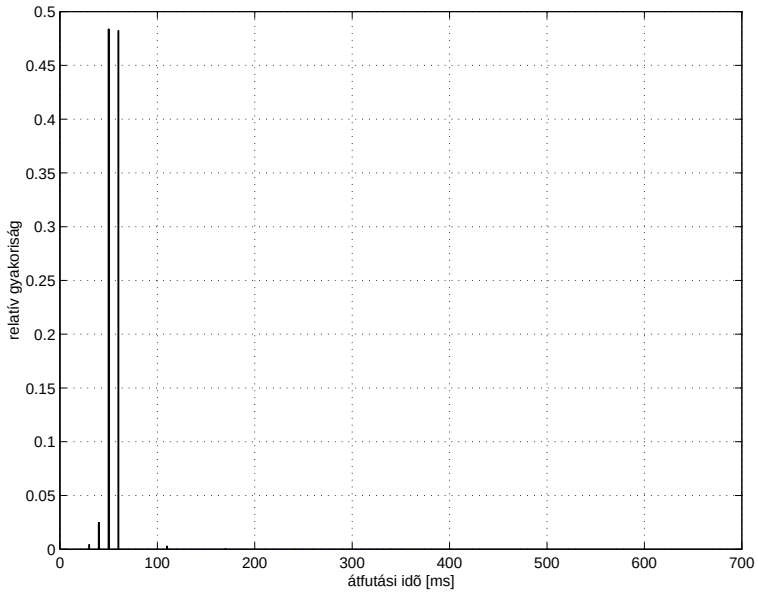
5.1 ábra Körülfordulási idő relatív gyakoriságfüggvény

szórással is rendelkezik. Ez ugyan nem látszik az ábrán, a mérés ugyanis viszonylag kis forgalmú órában történt, és a napi terhelésváltozás megfigyeléséhez – költségkímélés miatt – rövid ideig, de az nyilvánvaló tény, hogy a MikroWAPszerver esetében észlelhető késleltetést nem befolyásolják hasonló tényezők, az Interneten pedig napi viszonylatban is nagyot változik a forgalom, illetve a terheltség.

5.1.2. Teljes átfutási idő

A teljes átfutási idő annyiban különbözik a körülfordulási időtől, hogy az nem tartalmazza a két kommunikáló fél közötti adatút által okozott késleltetést. A diplomaterv keretén belül elkészített implementáció hatékonyságát tesztelendő, az átfutási időbe nem célszerű beleszámítani a hálózati réteg alatt elhelyezkedő (annak szolgálatot teljesítő) rétegek által okozott késleltetéseket, holott az egész rendszerre az mégis jellemző lenne. Az előzőek szerint az átfutási idő alatt a hálózati réteghez érkező enkapszulált kérés és az előállított válasz a hálózati rétegből történő kilépése között eltelt idő értendő. A számítógépen elvégzett mérések értékei kvantitatívan nem a mikrokontrollerbe beágyazandó rendszert jellemzik, de az egyes részkésleltetési idők aránya fontos a részfeladatok igényének arányaiban történő kifejezésére és az esetleges optimalizálás és egyszerűsítés céljának lokalizálásához. Természetesen a szabvány betartása a legfontosabb, de a hatékonyság fokozásának érdekében néhány jól megfontolt, a működést közvetlenül nem befolyásoló és indokolt egyszerűsítéssel élni lehet.

A fentiekben definiált átfutási idő értékeiből elállított relatív gyakoriságfüggvény az 5.2 ábrán látható. A MikroWAPszerverre mért körülfordulási időt és a hálózati réteg feletti



5.2 ábra Áfutási idő relatív gyakoriságfüggvény

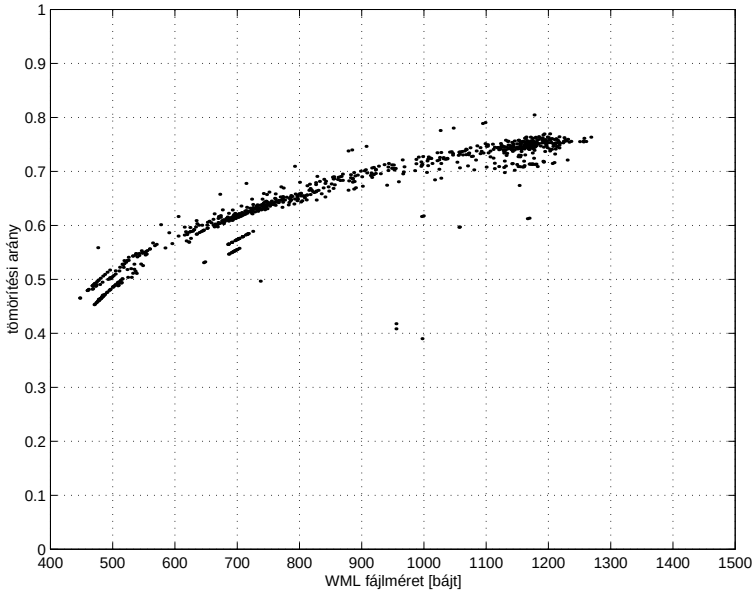
protokollrendszeren keresztüli teljes átfutási időt összehasonlítva látható, hogy a mikrokontrollerre való átültetés miatt megtöbbszöröződő átfutási idő esetén is elfogadható körülfordulási idő értékek kaphatók, hiszen a bemutatott paraméterek között nagyságrendnyi különbség van.

A teljes átfutási időhöz viszonyítva a következőkben ismertetésre kerülő részidőket, jól körvonalazható az egyes részfeladatok számítás- és időigénye. Ezek alapján esetlegesen felderíthetők az implementáció kevéssé hatékony részei is.

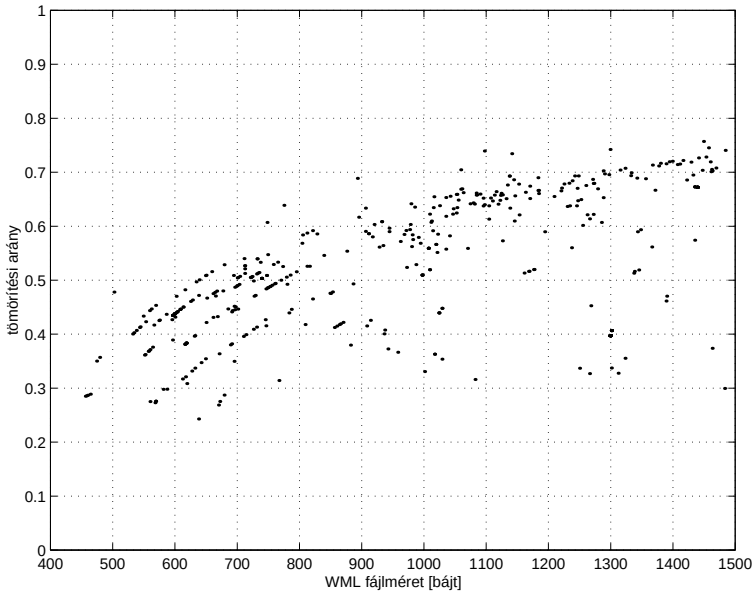
5.1.3. A WML fordító hatékonysága

A WML fordító több szempontból is kulcsfontosságú alkotóeleme a rendszernek. Célja, hogy a kis sebességű vonalon minimalizálja az átviendő adatmennyiséget, csökkentve ezzel az átvitelhez szükséges időt. Ehhez viszont egy processzorigényes fordítási feladatot kell elvégeznie. A mikrokontroller feldolgozási képessége töredéke a nagyteljesítményű számítógépekének, ezért amennyi idő nyerhető a kisebb mennyiségű adat átvitelével, annyi vagy akár több könnyen el is veszhet a fordítás során, ezért felvetődhet a kérdés, hogy miért van rá egyáltalán szükség. A fordító nem hagyható ki a rendszerből, két okból kifolyólag sem. A kliens csak a lefordított tartalmat, a bináris WML-t tudja csak értelmezni, valamint az alkalmazásfejlesztés gyakorlatilag lehetetlen lenne, ha a dinamikusan generálandó tartalmat a fejlesztőnek bináris formában kellene előállítania, illetve letárolnia a kiszolgálón. A fordítóra tehát mindenképpen szükség van, hatékonysága pedig az egész rendszer teljesítményére hatással van, ezért fontos ismerni, hogy milyen jellemzőkkel rendelkezik. A megítéléséhez két szempont is figyelembe veendő: milyen nyereséggel fordít, illetve mennyi időt vesz igénybe a fordítási művelet.

Az 5.3 ábrán két WAP portálról letöltött nagymennyiségű WML oldalra a MikroWAP-szerver fordítójával elvégzett fordítási művelet tömörítési hatékonyságára vonatkozó diagramok láthatók. Az eredeti, tömörítetlen (WML) állományok méretének függvényében vannak ábrázolva a tömörített és a tömörítetlen méret hányadosaként definiált tömörítési arányok. A nagymennyiségű eredménynek köszönhetően jól látható mindkét ábrán, hogy a pontok egymástól elkülönülő görbéket adnak. Az egyes görbék a közel azonos számú tag-et tartalmazó oldalak fordítási hatékonyságának jellegére utalnak. A tartalomszolgáltatók leginkább adatbázisból vett cikkek, hírek stb. felhasználásával dinamikusan állítják elő oldalaikat. Az adatbázisban természetesen formázó elemektől (tag-ektől) mentes tiszta, szöveges tartalmi információt tárolnak, hogy az több platformon is megfelelően felhasználható legyen. Ahhoz, hogy egy adatbázis rekord (cikk) WAP böngészésre alkalmas mobiltelefonon megjeleníthető legyen, WML oldalba kell ágyazni, azaz a cikk köré egy WML keretet kell illeszteni. Ezt a feladatot többféle szerveroldali technológiával is meg lehet oldani. A kérések hatására tehát a különböző témakörökhöz általában különböző, de adott témakörön belül nagy valószínűséggel azonos WML formázó eleme-



(a) http://www.port.hu/wap/



(b) http://wp.hu/

5.3 ábra Tömörítési arány az eredeti fájlméret függvényében

ket tartalmazó kerettel ellátott, változó tartalmú oldalak érkeznek. Ennek következtében tehát egy-egy cikkcsoport különböző hosszúságú híreihez ugyanazok a tag-ek adódnak. A WML fordító csak a tag-eket, illetve azok attribútumait tudja tömöríteni, a szöveges tartalmi információt nem. Így az azonos kerettel rendelkező oldalak tömörítésére nem a tömörítési arány, hanem az eredeti és tömörített fájlméret közötti különbség a konstans. A görbék íveltségét is ez okozza, ugyanis az ugyanazt a keretet tartalmazó oldalakra a tömörítési arány a

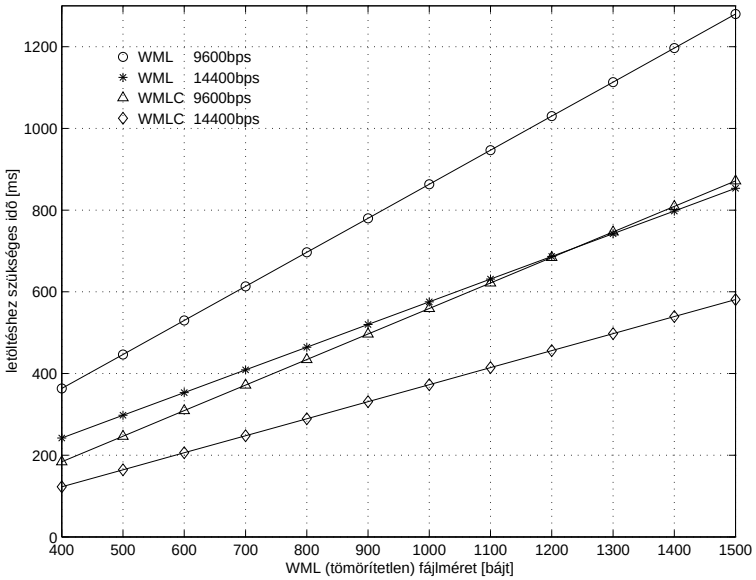
$$C_r = \frac{S_c}{S_o} = \frac{S_o - (S_o - S_c)}{S_o} \tag{5.1}$$

képlet alapján határozható meg, ahol C_r a tömörítési arány *(compression ratio)*, S_o az eredeti méret *(original size)* és S_c a tömörített fájlméret *(compressed size)*. Ha a $d = S_o - S_c$ különbség konstans, akkor a függvény grafikonja a

$$C_r = 1 - \frac{d}{S_o} \tag{5.2}$$

egyenlettel leírható hiperbola. Az 5.3(a) és az 5.3(b) ábrákon bemutatott grafikonokon kirajzolódó görbék tehát a különböző tag-számhoz tartozó hiperbolák részletei. Azt a következtetést lehet tehát levonni, hogy mindkét portál fejlesztői használták a dinamikus tartalom-generálás adta lehetőségeket. Az 5.3(b) grafikon azonban – különösen a másikhoz viszonyítva – meglehetősen nagy szórást mutat, amelynek a statikusan fejlesztett oldalak formázó és tartalmi elemeinek nagymértékben változó aránya az oka. Azonban túl a WML fordító hatékonyságának vizsgálatához letöltött teszttartalom fejlesztési technikáira tett becsléseken, az is elmondható, hogy – különösen kis méretű WML oldalak esetén – a fordítás jelentős mértékű méret- és letöltési idő csökkenést eredményez.

Az 5.4 ábra a különböző méretű szöveges- (WML), és – a fordítás során létrehozott – bináris állományok (WMLC) letöltéséhez szükséges számított, illetve becsült időket ábrázolja az eredeti fájlméret függvényében. A letöltési idő az átviendő adatmennyiségből és a csatorna adatátviteli sebességéből adódik. A GSM szabványos adatátviteli sebessége 9600 bit másodpercenként, de egyes szolgáltatók 14400bps-os sebességen is szolgáltatnak, ezért a diagram mindkét adatsebességre ábrázolja a letöltési időket, mind a tömörítetlen, szöveges, mind a lefordított, bináris állományokra vonatkoztatva. Fordítás nélkül valójában nem továbbítódik egyetlen dokumentum sem a kliens felé, csak a hatékonyság szemléltetése miatt vannak az ábrán feltüntetve az erre vonatkozó adatok is. A letöltési idő értékek természetesen a teljes, enkapszulált válaszra vonatkoznak, beleértve tehát a WSP-, az UDP- és az IP fejlécet, valamint a HDLC keretet is. Ez mindösszesen 36 bájt (HDLC – 4 bájt, IP fejléc – 20 bájt, UDP fejléc – 8 bájt, WSP fejléc – 4 bájt), átszámítva 288 bit többlet adat. A becslés során azt is figyelembe kell venni, hogy a nagyobb WML oldalak általában több tag-et tartalmaznak. A letöltött tesztoldalak tömörítetlen méreté-



5.4 ábra
Letöltési idő különböző sebességű csatornákon tömörített, illetve tömörítetlen állományokra

nek függvényében az eredeti és a tömörített fájlméretek közötti, bájtban mért különbség ($d = S_o - S_c$) egy lineáris trendvonallal becsülhető, melynek egyenlete:

$$d = 0.25 \cdot S_o + 115. \tag{5.3}$$

A különböző méretű WML oldalak letöltéséhez szükséges idő az (5.4), a lefordított, bináris fájlok letöltési ideje pedig az (5.3) felhasználásával az (5.5) alapján számítható.

$$t_{dl} = \frac{8\frac{bit}{byte} \cdot S_o + 288bit}{v_{dc}} \tag{5.4}$$

$$t_{dl} = \frac{8\frac{bit}{byte} \cdot (S_o - d) + 288bit}{v_{dc}} \tag{5.5}$$

A képletekben szereplő t_{dl} a letöltési idő *(downloading time)*, a v_{dc} az adathívás sebessége *(data call velocity)*, a 288 bit pedig az alsóbb rétegek fejléc-információi részére fenntartott hely, tehát az egyes idők a fejlécekkel növelt adatmennyiség letöltésére vonatkoznak. Jól látható, hogy nagymértékű gyorsítás érhető el az adatátvitelben a szöveges helyett tömörített állományokat továbbítva.

A WML fordító mind az 5.3, mind az 5.4 ábra szerint nagy hatékonysággal tömöríti a formázást szolgáló tag-eket és attribútumokat is szöveges formában tartalmazó WML oldalakat, és ezzel jelentős mértékben lecsökkenti a fájlok kis sávszélességű csatornán történő átviteléhez szükséges letöltési időt.

A mérések előtt felmerült az a kérdés, hogy – függetlenül attól, hogy architekturális kérdések miatt mindenképpen alkalmazni kell, – van-e haszna egyáltalán a WML fordítónak, tekintve a mikrokontroller kis számítási kapacitását. Az nyilvánvaló, hogy a protokollrendszer legnagyobb erőforrásigényét a fordító támasztja. Az elvégzett mérések is alátámasztották, hogy a fordítás teszi ki az átfutási idő mintegy 90 százalékát. Az átfutási idő értékelésénél azt állapítottuk meg, hogy annak a kisebb képességű architektúrára való átültetés következtében megtöbbszöröződött értéke is csak nagyon kis részét teszi ki a körülfordulási időnek, tehát amellett hogy a rendszer felépítését tekintve a fordító a protokollrendszer szerves része, és nem hagyható el, használata célszerű és gazdaságos.

5.1.4. Ellenőrzőösszeg számítás

Az internet modul egy csomag érkezésekor ellenőrzi a másik fél által a fejlécre számított ellenőrzőösszeget, valamint csomag küldése esetén biztosítva, hogy a címzett is ellenőrizhesse a csomagot, maga is kiszámítja a fejléc ellenőrzőösszegét. A szabvány alapján az IP-nek minden esetben ki kell számítania, illetve küldés esetén ellenőriznie kell a fejléc integritását biztosító összeget. Tekintve hogy míg a csomag a hálózati réteghez érkezik, egy ellenőrzőösszeget már validálni kellett a teljes adatra vonatkozólag (HDLC – High Level Data Link Control keretezés), tehát gyakorlatilag nem lehetséges, hogy a hálózati szintű csomag megsérüljön, az internet modulban levő ellenőrzési fázis, ha az a hatékonyság növeléséhez jelentős mértékben hozzájárul, a szabvány előírásával ellenkezőleg, a feladat érzékenységét tekintve elhagyható. Semmiképpen nem hagyható ki azonban a csomagok küldésénél az összeg kiszámítása, hiszen annak hiányában a másik fél (a mobiltelefon) – feltéve, hogy az előzőhöz hasonló meggondolások miatt abban egyáltalán implementálva van az ellenőrzés – hibásnak találná az érkezett datagramot.

A szállítási réteg szintén tartalmaz az adatintegritás megőrzését szolgáló ellenőrzőösszeg számítási és ellenőrzési részeket. Az UDP esetében az IP-nél tett megfontolások alapján ugyancsak alkalmazható – a teljesítményt fokozandó – az összeg figyelmen kívül hagyása. Az UDP specifikációja azonban lehetővé teszi a küldött csomagok esetén is az összeg kiszámításának mellőzését. Ebben az esetben az ellenőrzőösszeg helyére nulla illesztendő. Ha az érkezett csomaghoz tartozó ellenőrzőösszeg nulla, akkor azt a csomagot nem kell ellenőrizni. Figyelembe véve, hogy az UDP-hez történő érkezés előtt legalább egyszer ellenőrzésre kerül az adat sértetlensége, az UDP ellenőrzőösszeg számítására készült rutinok elhagyásával tovább növelhető lenne az implementáció hatékonysága.

Mivel például a WML fordítás időigényéhez képest – az elvégzett mérések szerint – az ellenőrzőösszeg kiszámításához szükséges idő még az UDP esetében is, ahol nem csak a fejlécre, hanem a teljes adatra kell azt kiszámítani, elhanyagolható (több nagyságrenddel kisebb), nincs különösebb értelme a számítási műveletek elhagyásának, így az implementáció sem tér el a szabvány előírásaitól.

5.2. Biztonság

Ha egy eszköz alkalmazható ipari szabályozási körök működésének figyelemmel kíséréséhez, és beavatkozásához, akkor fontos kérdés az ehhez alkalmazott eszköz stabilitása és biztonsága. Fontos, hogy illetéktelenek ne férhessenek hozzá, és akik hozzáféréssel rendelkeznek, rosszindulatúan ne tehessék azt működésképtelenné, vagy hibásan működővé. Ezek a kérdések természetesen csak a távoli elérésre vonatkoznak, az eszköz fizikai védelme nem tartozik ide, mert az ellen az implementáció szintjén nem lehet védekezni, hogy illetéktelenek fizikai kárt tegyenek a készülékben.

5.2.1. Hozzáférhetőség

Az adatkapcsolatot biztosító protokoll, a PPP specifikáció szerint tartalmaz autentikációs elemeket, az adatkapcsolat létrehozása tehát csak a megfelelő felhasználónév és jelszó ismeretében lehetséges. A biztonság tovább fokozható az alkalmazási réteg szintjén elhelyezhető hozzáférés-jogosultság ellenőrzéssel, amelyet a fejlesztő saját maga implementálhat a kész szerverhez járulékos alkalmazásként.

5.2.2. Stabilitás

Különösen fontos kérdés az implementáció stabilitása, ha az eszköz például egy szabályozási körben kerül alkalmazásra. Kívülről ugyan nem látható, de mind a statikus tartalom visszaadását, mind az azok dinamikus előállítását, C függvények végzik. A távvezérlés és távlekérdezés funkció megvalósítása csak dinamikusan generált oldalak, azaz az említett függvények paraméterezett meghívásával történhet. A hibás paraméterek megfelelő lekezelése az alkalmazásfejlesztő feladata, arról az alkalmazási rétegben a vezérlést végző függvényen belül kell gondoskodni.

A szolgálat megtagadását (DOS – Denial Of Service) célzó támadások ellen nincs implementálva védekezési mechanizmus, így egy számítógéppel és egy modemmel, a MikroWAPszerverhez tartozó mobil terminál hívószámának és az adatkapcsolat felépítéséhez szükséges felhasználói név és jelszó ismeretében, a MikroWAPszerver térdre kényszeríthető. Ehhez azonban az előbb felsorolt hozzáférési kódokra van szükség, azok bizalmas kezelése pedig az üzemeltető felelőssége. Egyes GSM telefonok rendelkeznek olyan funkcióval, amelynek segítségével lekérdezhető a hívó fél telefonszáma a soros interfészen keresztül. Ennek a lehetőségnek a kihasználásával a mobiltelefon interfészét kezelő rétegben könnyedén implementálható olyan programrész, amelynek segítségével, csak megadott telefonszámokról érkező hívásra válaszol a MikroWAPszerver. Ezzel megakadályozható az, hogy rosszindulatúan bárki lefoglalhassa az egyetlen csatornát, gátolva a jogosult felhasználó hozzáférését. Mindezek mellett a teljes stabilitás érdekében az alsóbb rétegekben meghatározható maximális kérés csomagméret, illetve az egymást követő ké-

rések közötti minimális várakozási idő, amelyek átlépése esetén az adatok feldolgozása nem szükséges.

5.3. Felhasználási kör

A MikroWAPszerver, képességei és a benne rejlő lehetőségek alapján igen sokrétűen alkalmazható, kifejlesztésének célja elsősorban a távolról is (mobilan) menedzselhető hardvervezérlés lehetőségének megteremtése volt. Felhasználható tehát minden olyan területen, ahol kapcsolat létesíthető a soros interfész, vagy az I/O csatornák felhasználásával, és a GSM hálózati lefedettség biztosítja a mobil terminál elérhetőségét. Így alkalmazható különböző, folyamatos felügyeletet nem igénylő vezérlési feladatokra szabályozási körökben, illetve bizonyos érzékelők állapotának lekérdezéséhez, valamint a két lehetőséget együtt alkalmazva érzékelőként és beavatkozóként egyszerre. Továbbfejlesztett változata egy olyan szabályozó lehetne, amely bizonyos algoritmus alapján, emberi beavatkozás nélkül működik, és közben távolról monitorozható, illetve szükség esetén vezérelhető. Egy megfelelő illesztő-, illetve vezérlőegységgel kiegészítve alkalmazható az úgynevezett intelligens ház vezérlési feladatainak ellátására, amelynek keretein belül megvalósítható például a házban elhelyezett riasztó állapotának, a pillanatnyi elektromos teljesítményfelvételnek, a szoba hőmérsékletének, a gáz-, villany-, és vízóra állapotának lekérdezése, illetve vezérlési funkcióként a garázsajtó nyitása, a fűtés szabályozása, kerti locsolórendszer beindítása stb.

5.4. Összegzés

Az elvégzett mérések és elemzések alapján a MikroWAPszerver implementált rétegei, hatékonysági szempontokat figyelembe véve megfelelnek az előzetes becsléseknek és előírásoknak. A protokollrendszeren történő keresztülfutás ideje töredéke a teljes körülfordulási időnek. Az átfutási idő legnagyobb részét a szöveges dokumentumok binárissá történő fordítása teszi ki, de ezzel a lépéssel a letöltési idő a kis átviteli sebességű csatornán nagymértékben csökkenthető, úgyhogy a fordítással elvett idő többszörösen visszatérül az átvitel során. A gépi kódú program mérete akkora, hogy a még kisebb utasításkészlettel rendelkező mikroszámítógép programtárában is maradéktalanul elfér, sőt további alkalmazások tárolására is van lehetőség. A futás közben használt memória pedig a WAP specifikációjában rögzített maximális csomagméretek és a hatékony programozás eredményeként jóval kisebb, mint amekkorával a mikrokiszolgáló hardveréül tervezett eszköz rendelkezik. További elemzések során az is bebizonyosodott, hogy a MikroWAPszerver sokrétű és hasznos feladatok ellátására alkalmas a biztonságosságra és a stabilitásra vonatkozó kompromisszumok nélkül.

6. fejezet

Továbbfejlesztési lehetőségek

A MikroWAPszerver jelenlegi készültségi állapotában is sokrétűen alkalmazható, azonban számos továbbfejlesztési lehetőség kínálkozik, amelyekkel bővítve nagyobb funkcionalitás érhető el, illetve szélesebb körben és könnyebben lehetne felhasználni.

6.1. Alkalmazáscímzés kiszolgálónév alapján

A MikroWAPszerver alkalmazási rétegének implementációja nem tesz különbséget a kérésekben szereplő különböző hosztnevek között. A kiszolgáló ugyanazt a választ adja a `http://microwapserver/index.wml` URL-lel megadott kérésre, mint például a `http://www.wap.hu/index.wml`-re. A megkülönböztetésre nem volt szükség, mert a kiszolgáló nem csatlakozik az Internethez, elsődlegesen nem átjáróként működik, nem kell tehát a megadott címről lekérnie az ott tárolt tartalmat, hanem saját magának kell azt előállítania. A megkülönböztetésnek azonban mégis lehet értelme, ha az egyes címekhez különböző alkalmazások vannak társítva. Így nem a teljes URL legvégén elhelyezkedő erőforrás azonosító (például index.wml) különbözteti meg az alkalmazást, hanem az URL hosztcím része. Ezt a lehetőséget kihasználva más alkalmazás társítható például a `http://futesszabalyozas/index.wml` címhez és más a `http://riaszto/index.wml`-hez.

6.2. Dinamikus képgenerálás

A WAP képformátuma, a WBMP (Wireless BitMaP) nem alkalmas fényképek, vagy hasonló grafikák ábrázolására a mobiltelefonok meglehetősen kicsi kijelzőjének köszönhetően, azonban kiválóan felhasználható mérésadatgyűjtés eredményének grafikus megjelenítésére. Ennek megvalósításához mindössze néhány rajzolófüggvény implementálása szükséges, amelyek meghatározott alakzatokat, illetve számadatok alapján dinamikusan generált grafikonokat állítanak elő.

6.3. Távoli tartalomfeltöltés

Egy új, meglehetősen egyszerű protokoll bevezetésével, illetve az UDP kihasználásával megvalósítható, hogy a felhasználó a mikrokiszolgálóra tartalmat tölthessen fel. Ehhez egy, például MS Windows platformon futtatható programra és egy modemre van szükség. A mikrokontroller képes saját programtárát írni, így távolról is van lehetőség a tartalom elmentésére.

6.4. Nyelvértelmező (interpreter)

A MikroWAPszerver programtárában természetesen a lefordított program van eltárolva. A felhasználó által feltöltendő CGI-képes dokumentumok nem kerülhetnek be a programtárba lefordított változatban, hiszen ehhez minden egyes felhasználónak meg kellene vásárolnia az adott mikrokontroller fejlesztői környezetetét és fordítóprogramját. Ezért szükséges egy úgynevezett nyelvértelmező vagy *interpreter*, amely képes feldolgozni és végrehajtani a meghatározott szintaktikájú programkódot.

6.5. Szerveroldali push alkalmazása

A szerveroldalon inicializált *push* metódus segítségével kliensoldali kérés nélkül küldhető adat a kliens felé. A WSP definiálja ezt a metódust, de a diplomaterv keretein belül nem került implementálásra. Olyan esetekben lehet például kihasználni, amikor folyamatos tájékoztatás szükséges egy eszköz állapotáról, úgy hogy akár percenként le kellene azt kérdezni.

7. fejezet

Összefoglalás

Egyre több, különböző képességű mikrokiszolgáló kapható a kereskedelemi forgalomban, amelyek más-más feladatok megoldására alkalmasak. Ezen szerverek nagytöbbsége csak a World-Wide Web technológiát támogatja, egy részük azonban képes WAP tartalom szolgáltatására is, de azon túl, hogy ismerik az egyes WAP formátumok MIME (Multipurpose Internet Mail Extensions) típusát, és a kérésre adott válaszban megadják azt, nem realizálják a protokollrendszer egyetlen elemét sem, ezért közvetlenül nem alkalmazhatók WAP-os mobiltelefon kiszolgálására. Ezek az eszközök is használhatók távvezérléshez, viszont mind szerver-, mind kliensoldalon helyhez kötöttek. A kötöttséget az internetelérési pont szükségessége okozza, a kliens felől azonban annyiban mobil a hozzáférés, hogy az Internetről bárhonnan elérhető. A mobilitás kliens- és szerveroldalra történő kiterjesztését teszi lehetővé a MikroWAPszerver nevet viselő miniatűr kiszolgáló. A diplomatervezés keretén belül, a feladatkiíráson túlmenően, a MikroWAPszerver hálózati, szállítási, viszony- és alkalmazási rétegei kerültek részletes tervezés után megvalósításra, figyelembe véve az implementáció célhardverének, egy mikroszámítógépnek a rendelkezésre álló szerény erőforrásait. Az elkészített implementáció a lehetőségekhez mérten a legnagyobb hatékonysággal valósítja meg a specifikált feladatokat. A jól meghatározott interfészeknek köszönhetően az egyes, külön modulokként fejlesztett rétegek összeillesztése különösebb nehézségek nélkül történhetett meg. A MikroWAPszerver a megvalósított protokollkészlettel teljes funkcionalitással képes kiszolgálni egy, a WAP szabványt maradéktalanul megvalósító mobiltelefont, vagy más mobil számítástechnikai eszközt. Felhasználva az előre elkészített, a tartalomfejlesztést segítő függvényeket, könnyen megvalósítható a tartalom dinamikus előállítása. A dinamikus tartalom-generálás függvényeit kiegészítve, egyszerűen oldható meg a MikroWAPszerverre csatlakoztatható külső eszközök távolról történő vezérlése és lekérdezése, amelyeket a kifejlesztett példaalkalmazás, a kiszolgálóhoz soros interfészen keresztül illesztett többfunkciós próbaáramkör vezérlési példája is illusztrál. Megfelelő illesztés segítségével tehát bonyolult vezérlőáramkörök irányíthatók és monitorozhatók.

Az implementáció hatékonyságát vizsgáló mérések arra utalnak, hogy a korlátozot-

tabb erőforrásokkal rendelkező mikroszámítógépre történő átültetés következtében megnövekedő feldolgozási idők sem befolyásolják jelentős mértékben a kliens felől érzékelhető, leginkább a kis átviteli sebességű csatornának köszönhető késleltetést, ezért az új architektúrán futtatva a MikroWAPszerver megvalósított rétegeit, közel azonos letöltési idők adódnak.

A hálózati és a felette elhelyezkedő rétegeket megvalósító, a diplomatervezés során elkészített implementáció nem tartalmaz hardver-specifikus elemeket és különböző architektúrán nem, vagy eltérően működő könyvtári függvényeket, ezért lényegében forrásszinten hordozható. A rétegek párhuzamos futtatását megvalósító kvázi operációs rendszernek, illetve ütemezőnek azonban illeszkednie kell az adott környezethez.

A megvalósított példaalkalmazás is bemutatta a MikroWAPszerver legfontosabb lehetőségeit, azonban néhány továbbfejlesztési tervet megvalósítva még tovább szélesíthető a felhasználási területe.

A. Függelék

WSP kódok

A.1 táblázat A WSP fontosabb státuszkódjai

HTTP státusz kód	Magyarázat	Hozzárendelt szám
100	folytatás	0x10
200	OK, rendben	0x20
204	nincs tartalom	0x24
300	több lehetőség	0x30
400	rossz kérés	0x40
401	nem authorizált	0x41
403	hozzáférés megtagadva	0x42
404	nem található	0x44
500	belső szerver hiba	0x60
503	szolgáltatás nem elérhető	0x63
505	nem támogatott HTTP verzió	0x65

A.2 táblázat PDU típus hozzárendelések

PDU típus	Hozzárendelt szám
Reply	0x04
Push	0x06
Get	0x40
Head	0x42
Post	0x60

A.3 táblázat Tartalomtípus hozzárendelések

Tartalom típusa	Hozzárendelt szám
text/html	0x02
text/plain	0x03
text/vnd.wap.wml	0x08
text/vnd.wap.wmlscript	0x09
application/vnd.wap.wmlc	0x14
application/vnd.wap.wmlscriptc	0x15
image/vnd.wap.wbmp	0x21
application/xml	0x27
text/xml	0x28
application/vnd/wap/wbxml	0x29

B. Függelék

WBXML kódok

B.1 táblázat A WML tag-ekhez hozzárendelt kódok

Tag név	Kód	Tag név	Kód
a	1C	postfield	21
anchor	22	pre	1B
access	23	prev	32
b	24	onevent	33
big	25	optgroup	34
br	26	option	35
card	27	refresh	36
do	28	select	37
em	29	setvar	3E
fieldset	2A	small	38
go	2B	strong	39
head	2C	table	1F
i	2D	td	1D
img	2E	template	3B
input	2F	timer	3C
meta	30	tr	1E
noop	31	u	3D
p	20	wml	3F

B.2 táblázat A WML tag-ek attribútumaihoz rendelt kódok

Attr. név	Attr. érték	Kód
accept-charset		05
acceskey		5E
align		52
align	bottom	06
align	center	07
align	left	08
align	middle	09
align	right	0A
align	top	0B
alt		0C
cache-control	no-cache	64
class		54
columns		53
content		0D
domain		0F
emptyok	false	10
emptyok	true	11
enctype		5F
format		12
forua	false	56
forua	true	57
height		13
href		4A
href	http://	4B
href	https://	4C
hspace		14
http-equiv		5A
http-equiv	Content-Type	5B
http-equiv	Expires	5D
id		55
ivalue		15
iname		16
label		18
localsrc		19
maxlength		1A
method	get	1B
method	post	1C
mode	nowrap	1D
mode	wrap	1F

Attr. név	Attr. érték	Kód
multiple	false	1F
multiple	true	20
name		21
newcontext	false	22
newcontext	true	23
onenterbackward		25
onenterforward		26
onpick		24
ontimer		27
optional	true	28
optional	false	29
path		2A
scheme		2E
sendreferer	false	2F
sendreferer	true	30
size		31
src		32
src	http://	58
src	https://	59
ordered	true	33
ordered	false	34
tabindex		35
title		36
type		37
type	accept	38
type	delete	39
type	help	3A
type	password	3B
type	onpick	3C
type	ontimer	3F
type	options	45
type	prev	46
type	reset	47
type	text	48
value		4D
vspace		4E
width		4F
xml:lang		50
xml:space	preseverve	62
xml:space	default	63

B.3 táblázat A WML tag-ek attribútumértékeihez rendelt kódok

Attribútum érték kezdet	Kód
.com/	85
.edu/	86
.net/	87
.org/	88
accept	89
bottom	8A
clear	8B
delete	8C
help	8D
http://	8E
http://www.	8F
https://	90
https://www.	91
middle	93
nowrap	94
onenterbackward	96
onenterforward	97
onpick	95
ontimer	98
options	99
password	9A
reset	9B
text	9D
top	9E
unknown	9F
wrap	A0
Www.	A1

Köszönetnyilvánítás

Ezúton szeretnék köszönetet mondani mindazoknak, akik valamilyen formában hozzájárultak diplomatervem elkészüléséhez. Külön köszönet illeti menyasszonyomat, Tóth Krisztinát türelméért és támogatásáért, Egedi Ferencet ötleteiért, tanácsaiért és programozástechnikai segítségéért, Kubinszky Ferencet, Lázár Zoltánt és Balogh Lászlót a munkám folyamatos figyelemmel kíséréséért és a konzultációkért, valamint Eged Bertalant a MikroWAPszerver alapelvének kidolgozásáért és konzulensi munkájáért. Köszönet jár az ingyenes professzionális LaTeX dokumentumkészítő programrendszer alkotóinak is, akik lehetővé tették számomra, hogy költségek nélkül, legális szoftver felhasználásával készíthessem el diplomamunkámat.

Tárgymutató

Rövidítések

CGI	Common Gateway Interface
CRC	Cyclic Redundancy Check
CSD	Circuit-Switched Data
GSM	Global System for Mobile Communications
HDLC	High-level Data Link Control
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICI	Interface Control Information
IDU	Interface Data Unit
IP	Internet Protocol
ISO	International Standard Organization
LCP	Link Control Protocol
NCP	Network Control Ptotocol
OSI	Open System Interconnect
PDU	Protocol Data Unit
PPP	Point-to-Point Protocol
SDU	Service Data Unit
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
URL	Uniform Resource Locator
URI	Uniform Resource Identifier
WAE	Wireless Application Environment
WAP	Wireless Application Protocol
WDP	Wireless Datagram Protocol
WML	Wireless Markup Language
WMLScript	Wireless Markup Language Script
WSP	Wireless Session Protocol
WTLS	Wireless Transport Layer Security
WTP	Wireless Transaction Protocol
WWW	World-Wide Web

Irodalomjegyzék

[1] WAP Forum, *Wireless Application Protocol Architecture Specification*, 1998 April.
URL `http://www.wapforum.org`

[2] Vaughan Pratt, *Matchbox Webserver*, 1999.
URL `http://wearables.stanford.edu/`

[3] H. Shrikumar, *IPic – A Match Head Sized Web-Server*, 1999.
URL `http://www-ccs.cs.umass.edu/~shri/iPic.html`

[4] D.B.P., *WWW Server in a Chip*, 2000.
URL `http://zhengxi.chat.ru/wwwpic/source.htm`

[5] Fredric White, *Webace Server – World's Smallest Web Server*, 2000.
URL `http://world.std.com/~fwhite/ace/`

[6] Dallas Semiconductor Corp., *TINI: Tiny Internet Interface*, 2001.
URL `http://www.ibutton.com/TINI/`

[7] FlatStack Ltd., *The Smallest Universal Web Server for Embedded Systems*, 2000.
URL `http://www.flatstack.com`

[8] Lightner Engineering, *PicoWeb Server*, 2001.
URL `http://www.picoweb.net/prodpw1.html`

[9] NetMedia Inc., *World's Smallest Ethernet Web Server*, 2000.
URL `http://www.siteplayer.com/`

[10] Warren Webb, *Embed a Tiny Web Server for Less Than $20*, EDN Europe, 2000 November.

[11] WAP Forum, *Wireless Application Protocol Wireless Session Protocol Specification*, 1999 May.
URL `http://www.wapforum.org`

[12] Andrew S. Tannenbaum, *Számítógép-hálózatok*, Panem Kft., 1999.

[13] D. D. Clark, *RFC 817: Modularity and Efficiency in Protocol Implementation*, MIT Laboratory for Computer Science, 1982 July.

[14] W. Richard Stevens, *TCP/IP Illustrated, Volume 1 : The Protocols*, Addison–Wesley Professional Computing Series, Addison–Wesley Pub Co, 1994 January.

[15] R. T. Braden, *RFC 1122: Requirements for Internet Hosts — Communication Layers*, Internet Engineering Taskforce, 1989 October.

[16] T. J. Socolofsky, C. J. Kale, *RFC 1180: TCP/IP Tutorial*, Spider Systems Limited, 1991 January.

[17] J. Postel, *RFC 791: Internet Protocol*, USC/Information Sciences Institute, 1981 September.

[18] J. Reynolds, J. Postel, *RFC 1700: Assigned Numbers*, USC/Information Sciences Institute, 1994 October.

[19] R. T. Braden, D. A. Borman, C. Partridge, *RFC 1071: Computing the Internet Checksum*, USC/Information Sciences Institute, 1988 September.

[20] J. Postel, *RFC 793: Transmission Control Protocol*, USC/Information Sciences Institute, 1981 September.

[21] J. Postel, *RFC 768: User Datagram Protocol*, USC/Information Sciences Institute, 1980 August.

[22] WAP Forum, *Wireless Application Protocol Wireless Application Environment Overview*, 1999 June.
URL `http://www.wapforum.org`

[23] WAP Forum, *Wireless Application Protocol Wireless Datagram Protocol Specification*, 1999 May.
URL `http://www.wapforum.org`

[24] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners Lee, *RFC 2616: Hypertext Transfer Protocol — HTTP/1.1*, UC Irvine, Compaq/W3C, Compaq, W3C/MIT, Xerox, Microsoft, W3C/MIT, 1999 June.

[25] WAP Forum, *Wireless Application Protocol Wireless Markup Language Specification Version 1.1*, 1999 June.
URL `http://www.wapforum.org`

[26] WAP Forum, *Wireless Application Protocol Binary XML Content Format Version 1.1*, 1999 June.
URL `http://www.wapforum.org`

[27] Standard Performance Evaluation Corporation, *SPECweb99 Benchmark*, 1999. URL `http://www.spec.org/osg/web99/index.html`

[28] Dezső Tamás, Egedi Ferenc, *A Wireless Application Protocol Stack teljesítmény tesztelése*, Tudományos Diákköri Konferencia, Budapesti Műszaki és Gazdaságtudományi Egyetem, 2000. november 10.