13. Egy 1 mm széles lézersugárral becélozunk egy 1 mm széles detektort, ami tőlünk 100 méterre van egy háztetőn. Mekkora szögeltérése lehet (fokban) a lézersugárnak, hogy még eltalálja a detektort?

14. Egy egyszerű távbeszélőrendszer két helyi központból és egy olyan távhívó központból áll, amelyhez mindkét helyi központ egy 1 MHz-es duplex trönkön keresztül kapcsolódik. Egy átlagos telefonkészüléken egy 8 órás munkanapon 4 hívást bonyolítanak le. Egy hívás átlagosan 6 percig tart. A hívások 10%-a távhívás (tehát a távhívó központon keresztül zajlik le). Maximálisan hány telefont tud kezelni egy helyi központ, ha 4 kHz-es áramköröket feltételezünk?

15. Egy helyi telefontársaságnak 10 millió előfizetője van. Minden készüléke rézvezetékkel kapcsolódik a telefonközponthoz. A vezetékek átlagos hossza 10 km. Mennyit ér az előfizetői hurkokban található réz? Tegyük fel, hogy a vezetékek keresztmetszete kör, átmérője pedig 1 mm. A réz sűrűsége 9 g/cm^3, és kilogrammonként 3 dollárt adnak érte.

16. A nagy teljesítményű mikroprocesszorok ára annyira lecsökkent, hogy már megéri őket beépíteni a modemekbe. Hogyan befolyásolhatja ez a telefonvonal hibáinak kezelését?

17. A 2.19. ábrán látható modemhez hasonlóan egy másik modem csillagkép mintázatának adatpontjai a következők: (1, 1), (1, –1), (–1, 1), (–1, –1). Mekkora adatátviteli sebesség érhető el egy ilyen modemmel 1200 baud esetén?

18. A 2.19. ábrán látható modemhez hasonlóan egy másik modem csillagkép mintázatának adatpontjai a következők: (0, 1), (0, 2). Fázismodulációt vagy amplitúdómodulációt használ a modem?

19. Beleillik-e az FTTH a végközpontokból, távhívó központokból stb. álló távbeszélőrendszer modellbe, vagy alapjaiban meg kell változtatni a modellt? Indokoljuk meg a válaszunkat!

20. Az előfizetői oldalon a távbeszélőrendszer csillag topológiájú, azaz egy körzet összes előfizetői hurokja egy helyi központba fut be. Ezzel szemben a kábeltelevíziós hálózat egyetlen hosszú vezetékből áll, ami végigkígyózik a körzeten, és az összes házat érinti. Tegyük fel, hogy a jövőben a kábeltévés vezeték 10 Gb/s-os fényvezető szál lesz, szemben a mostani rézvezetékkel. Képes lenne-e ez a kábel a telefonhálózatot oly módon szimulálni, hogy mindenkinek saját vonala legyen a helyi központig? Ha igen, akkor hány egytelefonos házat lehetne felfűzni egy ilyen kábelre?

21. Egy kábeltelevíziós rendszerben 100 kereskedelmi csatorna van, és mindegyik felváltva sugároz filmeket és reklámokat. Melyikhez hasonlít ez jobban, a TDM-hez vagy az FDM-hez?

22. Miért 125 μs a PCM rendszer mintavételi periódusideje?

23. Hány százalékos rátartással rendelkezik egy T1 vivő; azaz mennyi jut valójában az 1,544 Mb/s-os adatátviteli sebességből a felhasználónak?

24. Hasonlítsuk össze annak a két 4 kHz-es zajmentes csatornának a maximális adatátviteli sebességét, ahol az egyik analóg kódolással mintánként 2 bitet továbbít, a másik pedig a T1 PCM rendszerét használja!

25. Ha egy T1 vivőfrekvenciás rendszerben csúszás van, és a vevő kiesik a szinkronból, akkor a keretek első bitje segítségével megpróbál újraszinkronizálódni. Átlagosan hány keretet kell megvizsgálnia az újraszinkronizálódáshoz, ha a sikertelen újraszinkronizálódás valószínűsége 0,001?

26. Mi a különbség – ha van ilyen – egy *modem* demodulátor része és egy *kodek* kódoló része között? (Végül is mindkettő analóg jeleket alakít át digitális jelekké.)

27. Analóg jeleket egy 4 kHz-es csatornán 125 μs-onként mintavételezve digitálisan továbbítunk. Másodpercenként hány bitet tudunk valójában elküldeni a következő kódolási eljárásokkal:

(a) CCITT 2,048 Mb/s-os szabvány.

(b) 4 bit relatív jelszintű DPCM.

(c) Deltamoduláció.

28. Egy A amplitúdójú szinusz jelet deltamodulációval x minta/s mintavételi frekvenciával kódolunk. A +1 kimenet $+A/8$ jelszintváltozásnak felel meg, a –1 kimenet pedig $-A/8$ jelszintváltozásnak. Mekkora az a legnagyobb mintavételi frekvencia, amely mellett még nem keletkezik halmozott hiba?

29. A SONET órajelének pontossága kb. 10^{-9}. Mennyi idő alatt csúszik el az óra egy bitidőnyit? Mi következik a számításokból?

30. A 2.32. ábrán látható OC-3 felhasználói adatsebesség 148,608 Mb/s. Mutassuk meg, hogyan jön ki ez az érték a SONET OC-3 paramétereiből!

31. Milyen sávszélesség áll a felhasználó rendelkezésére az OC-12c csatorna esetén?

32. Adott három n csomópontból álló csomagkapcsolt hálózat. Az első egy olyan csillag topológiájú hálózat, amelyben van egy központi kapcsoló. A második hálózat egy (kétirányú) gyűrű, míg a harmadik egy olyan teljesen összekapcsolt hálózat, amelyben minden csomópont minden csomóponttal össze van kötve. Me-

lyik a legjobb, az átlagos, illetve a legrosszabb átviteli útvonal az átlépések száma szempontjából?

33. Hasonlítsuk össze egy x bites, k darab csomóponton átjutott üzenet késleltetését egy vonalkapcsolt és egy (alig terhelt) csomagkapcsolt hálózatban! Ha a kapcsolatfelépítés ideje s másodperc, a terjedési idő csomópontonként d másodperc, a csomagméret p bit és az adatátviteli sebesség b b/s, akkor milyen feltételek esetén lesz a csomagkapcsolt hálózat késleltetése kisebb?

34. Tegyük fel, hogy x bitnyi felhasználói adatot csomagok sorozataként továbbítunk egy csomagkapcsolt hálózatban úgy, hogy a csomagok k darab csomóponton mennek keresztül. Tegyük fel továbbá, hogy minden csomag p adatbitet és h fejrészbitet tartalmaz, ahol $x >> p + h$. Az adatvonalak átviteli sebessége b b/s, a vonali késleltetés pedig elhanyagolható. A p mely értékénél lesz a teljes késleltetés a legkisebb?

35. Hány kereszteződése van a 2.39.(a) és 2.39.(b) ábrán látható kapcsolóknak? Hasonlítsuk össze ezt az értéket egy 16 × 16-os, egyfokozatú keresztrudas kapcsoló kereszteződéseinek számával!

36. A 2.39.(a) ábrán látható térosztásos kapcsolónál mennyi az a legkevesebb kapcsolat, amennyinél egy újabb kimenő hívás már blokkolódna?

37. A 2.39.(a) ábrán látható kapcsoló egy lehetséges változata az, amikor a négy darab négyes kapcsoló helyett két darab nyolcas kapcsolót veszünk (tehát $n = 4$ és helyett $n = 8$). Egy ilyen kapcsoló csökkentené a hardverköltségeket, mivel csak két koncentrátorra lenne szükség a bemeneten és a kimeneten. Mi a legerősebb ellenérv egy ilyen kapcsoló megépítése ellen?

38. Hány vonalat tud egy időosztásos kapcsoló kezelni, ha a RAM hozzáférési idő 50 ns?

39. Hány bites RAM-pufferre van szüksége egy időréscserélőnek, ha a bemeneti minták 10 bitesek, és 80 bemeneti vonal van?

40. Jelent-e az időosztásos kapcsoló egy minimális késleltetést az egyes kapcsolási szinteken? Ha igen, akkor mennyit?

41. Mennyi ideig tart egy A/4-es (21 cm × 30 cm) kép elfaxolása az ISDN B csatornán, ha a faxgép 118 pixel/cm felbontással digitalizál, és egy pixelt 4 biten ábrázol? A mai faxgépek ennél még a hagyományos telefonvonalakon is gyorsabbak. Hogyan lehetséges ez?

42. Adjuk meg az ISDN hálózatban használt NT12 berendezés előnyeit és hátrányait az NT1 és NT2 berendezésekkel szemben!

43. A 2.50.(a) ábrán cellák ütközését láthatjuk, amint egy banyan kapcsolón áthaladnak. Az ütközések az első és a második fokozatban történnek. Lehet-e ütközés a harmadik fokozatban? Ha igen, akkor milyen feltételek esetén?

44. A következő feladat megoldásához lépésről lépésre kell követnünk néhány cellát egy Batcher–banyan kapcsolóban. A 0-tól 3-ig terjedő bemeneti vonalakra négy cella érkezik. A cellák sorrendben a 3-as, 5-ös, 2-es és 1-es kimenetekre akarnak eljutni. Határozzuk meg a továbbjutó cellákat a Batcher-kapcsoló mind a hat szintjén, és a banyan kapcsoló mind a négy szintjén! Amelyik vonalon nincs cella, oda tegyünk egy „–" jelet!

45. Ismételjük meg az előző feladatot úgy, hogy a bemenetekre érkező cellák sorrendben az alábbi kimenetekre akarnak eljutni: (7, –, 6, –, 5, –, 4, –).

46. Egy ATM kapcsoló 1024 bemenettel és 1024 kimenettel rendelkezik. A vonalak 622 Mb/s-os SONET sebességgel rendelkeznek, tehát a felhasználó számára kb. 594 Mb/s-os átvitelt tesz lehetővé. Mekkora teljes sávszélességre van szüksége a kapcsolónak ahhoz, hogy ekkora terhelést kezelni tudjon? Másodpercenként hány cellát kell tudnia feldolgozni?

47. Egy tipikus, hatszögletű cellákból felépülő mobiltelefon-rendszerben tilos egy adott frekvenciát a szomszédos cellában újra felhasználni. Ha összesen 480 frekvencia áll a rendelkezésünkre, akkor mennyit használhatunk egy cellában?

48. Becsüljük meg azt, hogy nagyjából hány 100 m átmérőjű PCS cellára van szükség ahhoz, hogy San Franciscot (120 km^2) lefedjük!

49. Amikor mobiltelefonon beszélünk, és egyik cellából átlépünk egy másik cellába, akkor időnként hirtelen megszakad a vonal, még akkor is, ha egyébként az adó és vevő berendezések tökéletesen működnek. Vajon miért?

50. Az Iridium projekt 66 alacsony röppályás műholdja hat láncot alkot a Föld körül. Abban a magasságban, ahol tartózkodnak, a periódusidő 90 perc. Egy helyhez kötött földi adóállomás számára átlagosan mennyi ideig „látható" egy ilyen műhold?

3. Az adatkapcsolati réteg

Ebben a fejezetben a 2. – adatkapcsolati – réteg tervezését fogjuk tanulmányozni. A vizsgálódás tárgya az lesz, hogy hogyan lehet megbízható, hatékony kommunikációt megvalósítani két szomszédos gép között. A szomszédosságon azt értjük, hogy a két gép fizikailag össze van kötve egy olyan kommunikációs csatornával, amely elméletileg vezetékként működik (pl. koaxiális kábel vagy telefonvonal). Az alapvető tulajdonság, ami egy csatornát „vezetékszerűvé" tesz az, hogy a rajta továbbított bitek a küldés sorrendjében érkeznek meg.

Elsőre azt gondolná az ember, hogy ez a probléma olyan egyszerű, hogy nincs is mit tanulmányozni – az *A* gép csak a vezetékre teszi a biteket, a *B* pedig veszi azokat. Sajnos, a kommunikációs áramkörök időnként hibáznak, ráadásul véges az adatátviteli sebességük és nem nulla késleltetéssel továbbítják a biteket. Ezekből a korlátokból fontos következtetéseket lehet levonni az adatátvitel hatékonyságára vonatkozóan. Az alkalmazott kommunikációs protokolloknak figyelembe kell venniük az összes ilyen tényezőt. Ezek a protokollok képezik e fejezet tárgyát.

Az adatkapcsolati szinten előforduló fő tervezési szempontok megismerése után elkezdjük tanulmányozni a réteg protokolljait: megnézzük a hibák természetét, kialakulásuk okait és hogy hogyan lehet jelezni, illetve javítani őket. Ezután egy sor egyre növekvő bonyolultságú, több és több, az adatkapcsolati rétegben jelenlevő problémát megoldó protokollt vizsgálunk meg. Végül a fejezetet a protokollmodellezéssel és protokollhelyesség-vizsgálattal, valamint az adatkapcsolati protokollokra adott néhány példával zárjuk.

3.1. Az adatkapcsolati réteg tervezési szempontjai

Az adatkapcsolati rétegnek számos speciális funkciót kell megvalósítania: jól definiált szolgálati interfészt kell nyújtania a hálózati rétegnek, meg kell határoznia, hogy a fizikai rétegben továbbított bitek hogyan legyenek keretekbe csoportosítva, foglalkoznia kell az átviteli hibákkal, és szabályoznia kell a keretek forgalmát, hogy a gyors adók ne tudják elárasztani a lassú vevőket. A következő szakaszokban ezeket a szempontokat vesszük sorra.

3.1.1. A hálózati rétegnek nyújtott szolgálatok

Az adatkapcsolati réteg feladata az, hogy szolgálatokat nyújtson a hálózati rétegnek. A legfőbb szolgáltatás az adatok átvitele a küldő gép hálózati rétegétől a célgép hálózati rétegéig. A küldő gépen van egy funkcionális egység – nevezzük folyamatnak – a hálózati rétegben, amely átad néhány bitet az adatkapcsolati rétegnek, hogy az továbbítsa a célhoz. Az adatkapcsolati réteg feladata az, hogy továbbítsa a biteket a célgéphez, hogy ott át lehessen azokat adni a hálózati rétegnek, ahogyan a 3.1.(a) ábrán látható. A valódi átvitel a 3.1.(b) ábra szerint megy végbe, de egyszerűbb két adatkapcsolati rétegbeli folyamatot elképzelni, melyek adatkapcsolati protokollal kommunikálnak. Emiatt ebben a fejezetben hallgatólagosan a 3.1.(a) ábra szerinti modellt fogjuk használni.

Az adatkapcsolati réteget különféle szolgálatok megvalósítására készíthetik fel. A ténylegesen megvalósított szolgálatok rendszerről rendszerre változhatnak. Három ésszerű, általánosan megvalósított lehetőség:

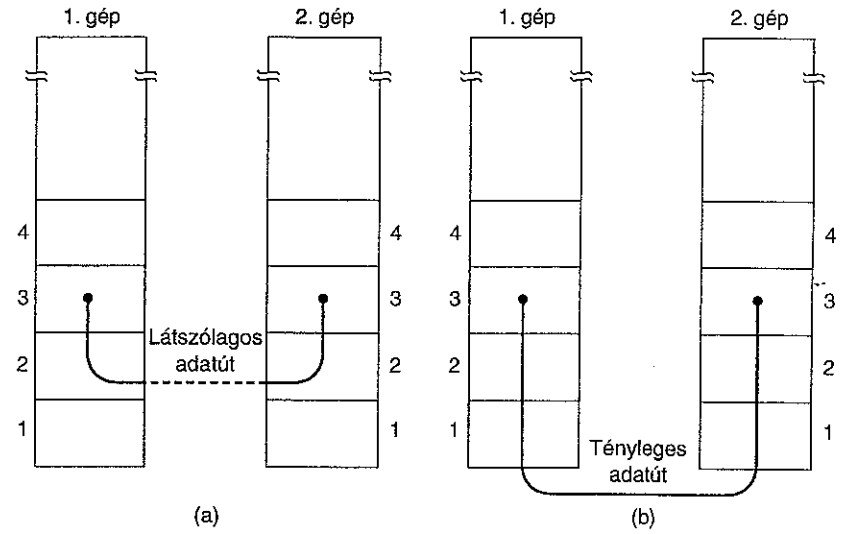
1. Nyugtázatlan összeköttetés nélküli szolgálat.

2. Nyugtázott összeköttetés nélküli szolgálat.

3. Nyugtázott összeköttetés alapú szolgálat.

Vizsgáljuk meg ezeket sorjában!

Nyugtázatlan összeköttetés nélküli szolgálat esetén a forrásgép egymástól függet-



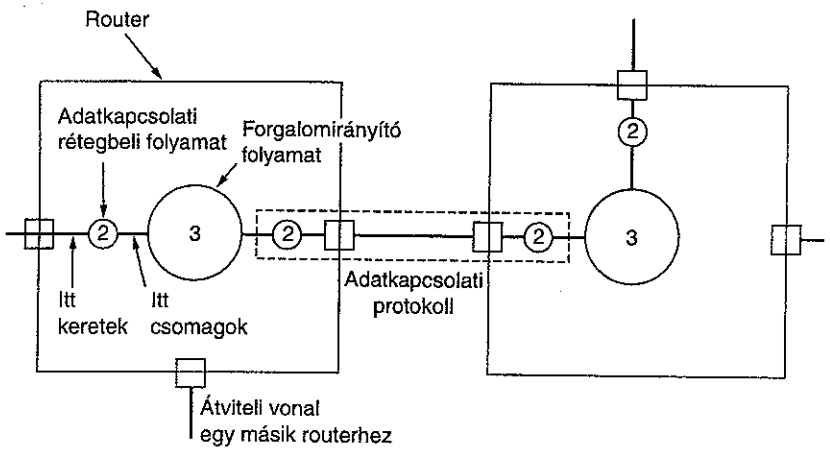
3.1. ábra. *(a) Látszólagos (virtuális) kommunikáció. (b) Tényleges kommunikáció*

len kereteket küld a célgép felé, amely nem nyugtázza a keretek megérkezését. Semmiféle kapcsolatot nem építenek fel előzetesen, illetve nem bontanak le az átvitel után. Ha egy keret a vonali zaj miatt elveszik, nem történik kísérlet a helyreállítására az adatkapcsolati rétegben. Ez a szolgálati osztály abban az esetben megfelelő, ha a hibaarány nagyon alacsony, így a hibák javítása a felsőbb rétegekre hagyható, valamint valós idejű forgalom esetén (pl. beszédátvitel), amikor a későn érkező adat rosszabb, mint a hibás adat. A legtöbb lokális hálózat nyugtázatlan összeköttetés nélküli szolgálatot alkalmaz az adatkapcsolati rétegben.

A következő lépés a megbízhatóság irányába a nyugtázott összeköttetés nélküli szolgálat. Ezen szolgáltatás esetén sincs felépített kapcsolat, de minden egyes elküldött keret megérkezését nyugtázza a célállomás, így a küldő értesül arról, hogy a keret megérkezett-e, vagy sem. Ha egy keret nem érkezik meg meghatározott időn belül, újra lehet küldeni. Ez a szolgálat megbízhatatlan csatornák (pl. vezeték nélküli rendszerek) esetén hasznos.

Talán érdemes hangsúlyozni, hogy a nyugtázás megvalósítása az adatkapcsolati rétegben sohasem elvárás, csak optimalizáció. A szállítási rétegben mindig meg lehet várni az elküldött üzenet nyugtázását. Ha a nyugta az időzítő lejárta előtt nem érkezik meg, a küldő újraküldheti az üzenetet. A probléma ezzel a stratégiával az, hogy ha az átlagos üzenet mondjuk 10 keretből áll, és általában a keretek 20 százaléka elveszik, nagyon sokáig tarthat, amíg az üzenet hibátlanul érkezik meg. Ha minden keretet egyenként nyugtázunk, és szükség esetén újraküldünk, sokkal gyorsabban jut át az egész üzenet. Megbízható csatornákon, mint például az üvegszál, a többletköltség miatt a bonyolult adatkapcsolati protokoll használata szükségtelen, de vezeték nélküli átviteli csatornákon a csatorna megbízhatatlansága miatt nagyon megéri.

Visszatérve a szolgálatokhoz, a legkifinomultabb szolgálat, amit az adatkapcsolati réteg a hálózati rétegnek nyújthat az összeköttetés alapú szolgálat. Ezt alkalmazva a forrás- és a célszámítógép felépít egy összeköttetést, mielőtt az adatátvitelt megkezde-



3.2. ábra. *Az adatkapcsolati protokoll elhelyezkedése*

nék. Minden elküldött keret sorszámozott, és az adatkapcsolati réteg garantálja, hogy a keretek valóban meg is érkezzenek, továbbá, hogy minden keret pontosan egyszer, és a megfelelő sorrendben érkezzen meg. Az összeköttetés nélküli szolgálatnál – ezzel ellentétben – elképzelhető, hogy egy elveszett nyugta a keret többszöri elküldését és így többszöri megérkezését okozza. Az összeköttetés alapú szolgálat megbízható keretfolyamot biztosít a hálózati réteg folyamatai számára.

Amikor összeköttetés alapú szolgálatot alkalmazunk, az átvitel három jól elkülöníthető fázisra bontható. Az első fázisban az összeköttetés felépül; mindkét oldalon inicializálódnak azok a változók és számlálók, melyek ahhoz szükségesek, hogy számon tarthassuk, hogy mely keretek érkeztek meg és melyek nem. A második fázisban egy vagy több keret tényleges továbbítása történik. A harmadik – egyben utolsó – fázisban az összeköttetést lebontjuk felszabadítva a változókat, pufferet és egyéb erőforrásokat, melyeket a kapcsolat karbantartásához használtunk.

Tekintsünk egy tipikus példát: egy WAN alhálózat routerekből és két pont közötti bérelt telefonvonalakból áll. Amikor egy keret megérkezik, a hardver megvizsgálja az ellenőrző összeget, majd továbbítja a keretet az adatkapcsolati réteg szoftverének (amely pl. egy chipbe lehet beágyazva a hálózatiadapter-kártyán). Az adatkapcsolati réteg szoftvere ellenőrzi, hogy ez-e a várt keret, és ha igen, a keret adat mezőjében levő csomagot átadja a router szoftvernek. Ez kiválasztja a megfelelő kimenő vonalat, és visszaadja a csomagot az adatkapcsolati rétegnek, amely elküldi azt. A két router közötti átvitel a 3.2. ábrán látható.

A forgalomirányító kód gyakran igényli, hogy a munka rendesen legyen elvégezve, azaz megbízható sorrendhelyes átvitelt feltételez minden két pont közötti (point-to-point) kapcsolaton, és nem szereti, ha gyakran kell foglalkozni az útközben elveszett csomagokkal. A szaggatott téglalapban látható adatkapcsolati protokoll feladata, hogy a megbízhatatlan kommunikációs vonalakat látszólag tökéletessé, vagy legalábbis elég jóvá tegye. Az adatkapcsolati protokoll ezen tulajdonsága különösen fontos vezeték nélküli kapcsolatok esetén, melyek természetüknél fogva nagyon megbízhatatlanok. Eltekintve attól, hogy az ábrán minden routerben több másolata látható az adatkapcsolati szoftvernek, egyetlen példány kezeli az összes vonalat, minden vonalhoz külön táblázatokkal és adatszerkezetekkel.

Bár ez a fejezet kimondottan az adatkapcsolati rétegről és az adatkapcsolati protokollokról szól, sok itt tanulmányozott elv, mint például a hibavédelem és a forgalomszabályozás, szintén megtalálható a szállítási és egyéb protokollokban.

3.1.2. Keretezés

Ahhoz, hogy az adatkapcsolati réteg szolgálatot nyújthasson a hálózati rétegnek, a fizikai réteg szolgálatát kell igénybe vennie. A fizikai réteg nem tesz mást, mint a kapott bitsorozatot megpróbálja továbbítani a célhoz. Az érkező bitsorozat hibamentességét a fizikai réteg nem garantálja. A bitek száma lehet kevesebb, azonos, vagy több mint az elküldötteké, és a bitek értéke is különbözhet az eredetitől. Az adatkapcsolati réteg feladata, hogy jelezze, illetve – ha szükséges – kijavítsa a hibákat.

A szokásos megoldás az, hogy az adatkapcsolati réteg különálló keretekre tördeli a

bitfolyamot, és minden kerethez kiszámolja az ellenőrző összeget. (Az ellenőrző-összeg-képző algoritmusokat a fejezet későbbi részében tárgyaljuk.) Amikor a keret megérkezik a célhoz, az ellenőrző összeget újra kiszámolja az adatkapcsolati réteg. Ha ez különbözik attól, amit a keret tartalmaz, a réteg tudja, hogy hiba történt, és lépéseket tesz ennek kezelésére (pl. eldobja a rossz keretet és hibajelzést küld vissza).

A bitfolyam keretekre tördelése sokkal bonyolultabb feladat, mint amilyennek első ránézésre tűnik. Egyik módja lehet a keretezésnek az, hogy szüneteket szúrunk be a bitfolyamba nagyon hasonlóan egy közönséges szövegben levő szavak közötti szóközökhöz. A hálózatok azonban ritkán nyújtanak garanciát az időzítéssel kapcsolatosan, így ezek a szünetek lerövidülhetnek, vagy újabb szünetek jelenhetnek meg az átvitel során.

Mivel túl kockázatos a keretek kezdetének és végének megjelölését az időzítésre bízni, más módszereket kellett kitalálni. Ebben a szakaszban négy módszert nézünk meg:

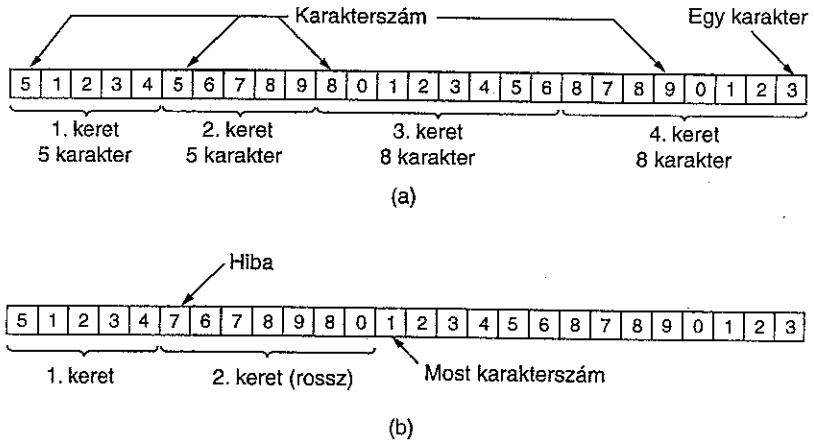
1. Karakterszámlálás.

2. Kezdő- és végkarakterek karakterbeszúrással.

3. Kezdő- és végjelek bitbeszúrással.

4. Fizikai rétegbeli kódolássértés.

Az első keretezési módszer a keretben levő karakterek számának megadására egy a keret fejrészében levő mezőt használ. Amikor a célállomás adatkapcsolati rétege megkapja a keretben levő karakterek számát, tudni fogja, hogy mennyi karakternek kell érkeznie, és így azt is, hogy hol van a keret vége. Ennek a technikának az alkalmazása a 3.3.(a) ábrán látható négy keretre, melyek mérete 5, 5, 8 és 8 karakter.



3.3. ábra. *Egy karakterfolyam. (a) Hiba nélkül. (b) Egy hibával*

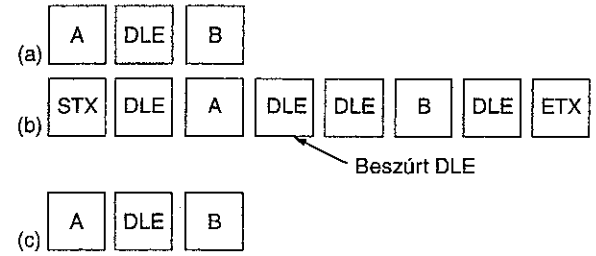
Ezzel az algoritmussal az a baj, hogy egy átviteli hiba elronthatja a karakterszám-mezőt. Például, ha egy 5-ös karakterszám a 3.3.(b) ábra második keretében 7-té válik, a célállomás kiesik a szinkronból, és képtelen lesz megtalálni a következő keret elejét. Még ha az ellenőrző összeg hibás, a cél tudja, hogy a keret rossz, akkor sincs mód arra, hogy megmondjuk, hol kezdődik a következő keret. Szintén nem segít a keret újraküldésének kérése, hiszen a cél nem tudja, hogy hány karaktert kell átlépnie ahhoz, hogy az újraküldött keret elejéhez érjen. A fentiek miatt a karakterszámlálásos módszert ma már ritkán használják.

A második keretezési módszer megoldja az újraszinkronizálás problémáját: minden keret a DLE STX ASCII karaktersorozattal kezdődik, és a DLE ETX-szel fejeződik be. (DLE: Data Link Escape – adatkapcsolati ESC karakter, STX: Start of TeXt – szöveg kezdete, ETX: End of TeXt – szöveg vége.) Így, ha a célállomás szem elől téveszti a kerethatárokat, nem kell mást tennie, mint a DLE STX vagy DLE ETX karaktereket figyelnie, és így újra megtalálja.

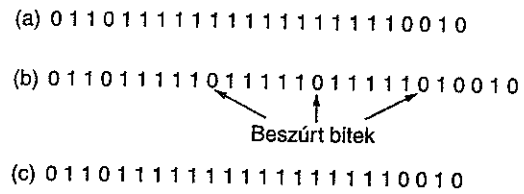
Komoly probléma jelentkezik ezzel a módszerrel bináris adatok (tárgykódú programok, lebegőpontos számok stb.) továbbításakor. Könnyen előfordulhat, hogy a DLE STX vagy a DLE ETX karaktersorozat megtalálható az adatok között, és így megzavarja a keretezést. Egyik módja a probléma megoldásának az, hogy a küldő adatkapcsolati rétege ASCII DLE karaktert szúr be minden, az adatfolyamban „véletlenül" előforduló DLE karakter elé. A vevő adatkapcsolati rétege eltávolítja a beszúrt DLE karaktereket, mielőtt az adatot továbbadná a hálózati rétegnek. Ezt a technikát **karakterbeszúrásnak (character stuffing)** nevezik. A keretező DLE STX vagy DLE ETX elkülöníthető az átviendő adatban levőktől az alapján, hogy a DLE egyedül áll-e, vagy sem. Az adatban levő DLE karakterek mindig meg vannak kettőzve. A 3.4. ábra egy példa adatfolyamot mutat beszúrás előtt, beszúrás után, és a beszúrt karakterek eltávolítása után.

A fő hátránya ennek a keretezési módszernek az, hogy szorosan kötődik a 8 bites, ezen belül az ASCII karakterkódokhoz. A hálózatok fejlődése során nyilvánvalóvá váltak a hátrányai annak, hogy a keretezés függjön a karakterkészlet kódolási eljárásától, így egy új módszert kellett kidolgozni, amely megengedi tetszőleges méretű (és kódolású) karakterek használatát.

Ez az új módszer lehetővé teszi, hogy tetszőleges számú bit legyen egy keretben, és az alkalmazott karakterkódok is tetszőleges számú bitet tartalmazzanak. A módszer az



3.4. ábra. *(a) A hálózati réteg által küldött adat. (b) A karakterbeszúrás utáni adat. (c) A vevőoldali hálózati rétegnek továbbadott adat*



3.5. ábra. *Bitbeszúrás. (a) Az eredeti adat. (b) Az átviteli vonalon megjelenő adat. (c) A vevő memóriájában megjelenő, a beszúrt bitek törlése utáni adat*

alábbiak szerint működik: minden keret egy speciális, **jelző (flag) bájtnak** nevezett bitmintával kezdődik. Ez a 01111110. Amikor az adó adatkapcsolati rétege öt egymást követő 1-es bitet talál az adatok között, automatikusan beszúr egy 0-t a kimenő bitfolyamba. Ez a **bitbeszúrás (bit stuffing)** analóg a karakterbeszúrással, amelyben egy DLE-t szúrtunk be a kimenő karakterfolyamba az adatok között levő DLE elé.

Amikor a vevő öt egymást követő 1-es bitet talál, melyet egy 0-s követ, automatikusan törli a 0-s bitet. Ahogyan a karakterbeszúrás teljesen átlátszó a hálózati réteg számára mindkét számítógépben, a bitbeszúrás is az. Ha a felhasználói adat tartalmazza a jelző bájt bitmintáját (01111110), ez 011111010-ként továbbítódik, de a vevő memóriájában már 01111110 jelenik meg. A 3.5. ábra egy példát mutat a bitbeszúrásra.

A bitbeszúrásos módszer segítségével egyértelműen felismerhetők a kerethatárok: ha a vevő szem elől téveszti a határokat, semmi mást nem kell tennie, mint a bemeneti bitfolyamban a jelző mintát keresnie, hiszen a minta csak kerethatárokon fordulhat elő, az adatok között sohasem.

Az utolsó keretezési eljárás olyan hálózatokban használható, ahol a fizikai rétegbeli kódolás redundanciát tartalmaz. Például néhány LAN egy adatbitet két fizikai szinten kódol: az 1-es bit egy fizikai magas-alacsony pár, a 0-s pedig egy alacsony-magas. A magas-magas és alacsony-alacsony kombinációk nem használatosak adatbitek kódolására (ezeket lehet a kerethatárok jelzésére használni). Az elgondolás azon alapul, hogy minden bit tartalmazzon átmenetet a közepénél, így a vevő könnyedén megtalálhatja a bithatárokat. Az érvénytelen fizikai kódok ilyetén alkalmazása része a 802-es LAN szabványnak, melyet a 4. fejezetben fogunk tanulmányozni.

Végül megjegyezzük, hogy sok adatkapcsolati protokoll a nagyobb biztonság érdekében a karakterszámlálás és valamelyik másik módszer kombinációját alkalmazza. Amikor egy keret megérkezik, a protokoll a karakterszámmező alapján keresi meg a keret végét. Csak akkor fogadja el a keretet érvényesnek, ha a megfelelő határoló jel a megfelelő helyen megvan, és az ellenőrző összeg is helyes. Ellenkező esetben a bemeneti adatfolyamban megkeresi a következő határoló jelet.

3.1.3. Hibavédelem

Miután megoldottuk a keretek kezdetének és végének jelzését, szembekerülünk a következő problémával: hogyan bizonyosodjunk meg arról, hogy minden keret ténylegesen megérkezik-e a célállomás hálózati rétegéhez, és hogy helyes sorrendben érkezik-e

meg. Tételezzük fel, hogy az adó folyamatosan bocsátotta ki a kereteket attól függetlenül, hogy azok megfelelően megérkeztek-e. Ez jó lehet a nyugtázatlan összeköttetés nélküli szolgálathoz, de bizonyára nem lenne jó a megbízható, összeköttetés alapú szolgálathoz.

A biztonságos átvitel megvalósításának általános módja az, hogy az adónak valamilyen visszacsatolást biztosítunk arról, hogy mi történik a vonal másik végén. Tipikusan az adó megköveteli a vevőtől, hogy speciális vezérlő kereteket küldjön vissza, melyek pozitív vagy negatív nyugtát hordoznak a bejövő keretekről. Ha a küldő pozitív nyugtát kap egy keretről, tudja, hogy a keret rendben megérkezett. A negatív nyugta ellenben azt jelenti, hogy valami nincs rendben, és a keretet újra kell adni.

További komplikáció származik abból, hogy hardver hibák (pl. zaj) okozhatják egy keret teljes eltűnését. Ebben az esetben az adó egyáltalán nem reagál, mivel nincs is mire reagálnia. Tisztán látszik, hogy az a protokoll, amelyben a küldő a keret elküldése után vár a pozitív vagy negatív nyugtára, örökre felfüggesztődne, ha egy keret egyszer teljesen elveszne a hibásan működő hardver miatt.

Ezt a lehetőséget az adatkapcsolati rétegben időzítők bevezetésével küszöbölik ki. Amikor az adó továbbít egy keretet, általában egy időzítőt is elindít. Az időzítő úgy van beállítva, hogy lejártáig legyen elég idő arra, hogy a keret elérje a célt, ott feldolgozásra kerüljön és a nyugta visszatérjen az adóhoz. Normális esetben a keret helyesen megérkezik, és a nyugta visszaér, mielőtt az időzítő lejárna. Ekkor az időzítő törlődik.

Ha viszont a keret vagy a nyugta elveszik, az időzítő lejár, és jelzi az adónak, hogy valószínűleg hiba történt. A nyilvánvaló megoldás: egyszerűen újra elküldeni a keretet. Ha azonban a kereteket többször továbbítjuk, fennáll a veszélye annak, hogy a vevő többször veszi ugyanazt a keretet, és többször adja át a hálózati rétegnek. Hogy ezt megakadályozzuk, általában a kimenő kereteknek sorszámot adunk, hogy a vevő meg tudja különböztetni az újraadott kereteket az eredetiektől.

Az adatkapcsolati réteg feladatának fontos része az, hogy az időzítőket és a számlálókat úgy kezelje, hogy biztosítani tudja, a keretek pontosan egyszeri (nem több és nem kevesebb) megérkezését a célállomás hálózati rétegéhez. E fejezet egy későbbi részében részletesen tanulmányozni fogjuk egy sor egyre bonyolultabb példán keresztül, hogy hogyan lehet ezt megvalósítani.

3.1.4. Forgalomszabályozás

Egy másik fontos tervezési kérdés, amely megjelenik az adatkapcsolati rétegben (és a felsőbb rétegekben is) az, hogy mit tegyünk azzal az állomással, amelyik rendszeresen gyorsabban akarja adni a kereteket, mint a vevő azokat fogadni tudná. Ez a szituáció könnyen előállhat akkor, ha az adó egy gyors (vagy kevéssé terhelt) számítógép, a vevő pedig egy lassú (vagy erősen leterhelt) gép. Az adó folyamatosan pumpálja kifelé a kereteket egészen addig, míg a vevőt teljesen el nem árasztja. Még ha az átvitel hibamentes is, a vevő egy bizonyos ponttól kezdve nem lesz képes kezelni a folyton érkező kereteket, és néhányat el fog veszíteni. Világos, hogy valamit tenni kell, hogy megakadályozzuk ennek kialakulását.

A szokásos megoldás a **forgalomszabályozás (flow control)** bevezetése, melynek segítségével az adót olyan mértékben visszafogjuk, hogy a vevő már tudja kezelni az általa generált forgalmat. Ez a lassítás általában valamilyen visszacsatolási mechanizmust igényel azért, hogy az adónak tudomására jusson, hogy a vevő tudja-e folytatni tevékenységét.

Különféle forgalomszabályozási elgondolások ismertek, de legtöbbjük ugyanazt az alapelvet alkalmazza. A protokoll jól definiált szabályokat tartalmaz arra vonatkozóan, hogy a következő keretet mikor küldheti el az adó állomás. Ezek a szabályok általában megtiltják egy keret elküldését, amíg a vevő – akár implicit, akár explicit módon – engedélyt nem ad rá. Például, amikor egy kapcsolat felépül, a vevő mondhatja a következőt: „Most küldhetsz nekem *n* keretet, de ha ezeket elküldted, ne küldj többet addig, amíg nem szólok." Ebben a fejezetben különféle forgalomszabályozási mechanizmusokat fogunk tanulmányozni, melyek ezen az elven alapulnak. A későbbi fejezetekben másféle mechanizmusokat is megvizsgálunk.

3.2. Hibajelzés és -javítás

Ahogyan a második fejezetben láttuk, a telefonrendszer három részből áll: a telefonközpontokból, a központok közötti trönkökből és az előfizetői hurkokból. Az első két rész ma már majdnem teljesen digitális az Egyesült Államokban és néhány más országban (pl. Magyarországon is – *a fordító megj.*). Az előfizetői hurkok még mindig analóg, csavart érpáros rézvezetékek, és a jövőben is azok maradnak lecserélésük hatalmas költsége miatt. A digitális részen a hibák ritkák, viszont az előfizetői hurkokon még mindennaposak. Ráadásul a vezeték nélküli kommunikáció egyre hétköznapibbá válik, és itt a hibaarányok nagyságrendekkel nagyobbak, mint a központok közötti üvegszálas trönkökön. Ebből az következik, hogy az átviteli hibákkal még sok éven át együtt kell élnünk.

Néhány átviteli közegre (pl. rádió) jellemző, hogy a hibák – az őket előidéző fizikai folyamatok természete miatt – jóval gyakrabban fordulnak elő csoportosan, mint egyesével. A csoportosan jövő hibáknak megvan az előnyük és a hátrányuk is a különálló, csupán egy bitet érintőekkel szemben. Az előny a következő: a számítógépes adatok bitjei mindig blokkokban kerülnek elküldésre. Tételezzük fel, hogy a blokkméret 1000 bit, és átlagosan 0,001 hiba van bitenként. Ha a hibák függetlenek lennének, a legtöbb blokk tartalmazna hibát. Ha a hibák 100-as csoportokban jönnek, 100 közül átlagosan csak egy vagy két blokkot érintenek. A hátránya a csoportos hibáknak az, hogy sokkal nehezebb jelezni és kijavítani őket, mint a különállóakat.

3.2.1. Hibajavító kódok

A hálózattervezők két alapvető stratégiát fejlesztettek ki a hibák kezelésére. Az egyik szerint annyi redundáns információt teszünk minden továbbítandó adatblokkhoz, hogy hiba esetén a vevő ki tudja következtetni, hogy minek kellene a továbbított karakternek lennie. A másik szerint csak annyi redundanciát teszünk az adatokhoz, hogy a vevő a hiba jelenlétét tudja kikövetkeztetni, de azt nem, hogy melyik, miben volt hibás. Ebben az esetben a vevő az adat újraküldését kéri. Az előbbi stratégia **hibajavító kódokat (error-correcting codes)**, az utóbbi **hibajelző kódokat (error-detecting codes)** használ.

Ahhoz, hogy megértsük a hibák kezelésének módját, közelebbről meg kell néznünk, hogy mi is egy hiba valójában. Rendszerint egy keret m adatbitből (ez az üzenet) és r redundáns vagy ellenőrző bitből áll. Legyen a teljes hossz n (azaz $n = m + r$). Az n bites, adat- és ellenőrző biteket tartalmazó egységre gyakran **kódszóként (codeword)** hivatkozunk.

Adott két tetszőleges kódszó, mondjuk az 10001001 és az 10110001. Meg tudjuk határozni, hogy hány, a két kódszóban azonos helyen levő bit értéke különbözik. Ebben az esetben 3 eltérő bit van. Ahhoz hogy az eltérő bitek számát meghatározzuk, csak KIZÁRÓ VAGY kapcsolatba kell hozni a kódszavakat, és meg kell számolni az eredményben levő 1-es biteket. Azoknak a bitpozícióknak a számát, melyekben a két kódszó különbözik, a kódszavak **Hamming távolságának** (Hamming, 1950) nevezzük. A fogalom jelentése más megközelítésből az, hogy ha két kódszó Hamming-távolsága d, akkor d darab egybites hiba kell ahhoz, hogy az egyik kódszót a másikba vigye.

A legtöbb adatátviteli alkalmazásban mind a 2^m lehetséges adatüzenet legális, de az ellenőrző bitek kiszámítási módja miatt nem fordul elő mind a 2^n lehetséges kódszó. Megadva az ellenőrző bitek kiszámításának módját, meg lehet alkotni a legális kódszavak teljes listáját, és ebből a listából ki lehet keresni azt a két kódszót, melyeknek legkisebb a Hamming-távolsága. Ez a távolság a teljes kód Hamming-távolsága.

Az hogy egy kód hibajelző vagy hibajavító tulajdonságú-e, a kód Hamming-távolságától függ. Ahhoz, hogy d hibát jelezni tudjunk, $d + 1$ Hamming-távolságú kód kell, mert egy ilyen kódban d bithiba nem tudja a kódszót egy másik érvényes kódszóba vinni. Amikor a vevő egy érvénytelen kódszót lát, tudja, hogy átviteli hiba történt. Hasonlóan: ahhoz, hogy d hibát ki tudjunk javítani, $2d + 1$ Hamming-távolságú kód kell, mert így az érvényes kódszavak olyan távol vannak, hogy még d bit megváltozásakor is közelebb lesz az eredeti kódszó a hibáshoz, mint bármely másik érvényes, így az egyértelműen meghatározható.

Egyszerű példaként a hibajelző kódra vegyünk egy kódot, melyben egy **paritásbitet (parity bit)** fűzünk az adatok végéhez. A paritásbitet úgy választjuk meg, hogy a kódszóban levő 1-ek száma páros (vagy páratlan) legyen. Például, ha az 10110101-et egy bit hozzáfűzésével páros paritással továbbítjuk, a kódja 101101011 lesz, míg az 10110001-ből 101100010 lesz páros paritással. Egy egy paritásbites kód Hamming-távolsága 2, hiszen bármilyen 1 bites hiba rossz paritású kódszót eredményez. Ezért az ilyen kód 1 bitnyi hiba jelzésére alkalmas.

Egyszerű példaként a hibajavító kódra, vegyünk egy kódot, ami csak négy érvényes kódszót tartalmaz:

0000000000, 0000011111, 1111100000 és 1111111111

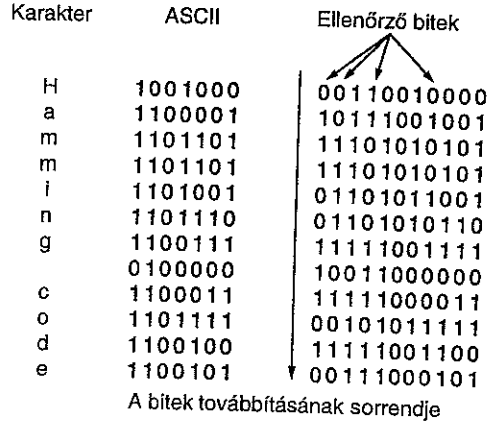
Ennek a kódnak a Hamming-távolsága 5, ami azt jelenti, hogy kétbitnyi hibát tud ja-

vítani. Ha a 0000000111 kódszó érkezik, a vevő tudja, hogy az eredetinek 0000011111-nek kellett lennie. Azonban, ha hárombitnyi hiba változtatta a 0000000000-t 0000000111-re, akkor a kódszó nem megfelelően lesz kijavítva.

Képzeljük el, hogy egy olyan kódot akarunk tervezni m üzenet- és r ellenőrző bittel, amely minden egybites hibát ki tud javítani. A 2^m érvényes üzenet mindegyikéhez van n darab, tőle 1 távolságra levő érvénytelen kódszó. Ezeket úgy kaphatjuk meg, hogy az üzenetből képzett n bites kódszó minden egyes bitjét egyenként invertáljuk. Így a 2^m érvényes üzenet mindegyikéhez $n + 1$ bitmintát kell hozzárendelnünk. Mivel a bitminták teljes száma 2^n, igaznak kell lennie az $(n + 1)2^m = 2^n$ egyenlőtlenségnek. Felhasználva, hogy $n = m + r$, a feltétel $(m + r + 1) \leq 2^r$. Megadva m-et kapunk egy alsó korlátot arra, hogy hány ellenőrző bit szükséges ahhoz, hogy az egybites hibákat ki tudjuk javítani.

Az elméleti alsó korlát valóban elérhető a Hammingnek (1950) köszönhető eljárással. A kódszó bitjeit balról jobbra, 1-gyel kezdődően megszámozzuk. Azok a bitek, melyek sorszáma 2 egész számú hatványa (pl. 1, 2, 4, 8, 16 stb.), ellenőrző bitek. A maradék bithelyeket (3, 5, 6, 7, 9 stb.) az üzenet bitjeivel töltjük ki. Mindegyik ellenőrző bit a bitek valamilyen csoportjának (beleértve önmagát is) a paritását állítja be párosra (vagy páratlanra). Egy bit számos paritásszámítási csoportba tartozhat. Ahhoz hogy megállapítsuk, hogy a k-ik pozícióban levő adatbit melyik ellenőrző bit kiszámításában vesz részt, írjuk fel k-t 2 hatványainak összegeként. Például $11 = 1 + 2 + 8$ és $29 = 1 + 4 + 8 + 16$. Egy bitet azok a paritásbitek ellenőriznek, amelyek sorszáma szerepel az így képzett összeg tagjai között (pl. a 11. bitet az 1., 2. és a 8. bit ellenőrzi).

Amikor a kódszó megérkezik, a vevő nullára állít egy számlálót. Ezután megvizsgál minden ellenőrző bitet (1., 2., 4., 8., ...), hogy a paritása jó-e. Ha a k-ik paritásbit nem jó, hozzáadja k-t a számlálóhoz. Ha a számláló 0 az összes ellenőrző bit megvizsgálása után (azaz mindegyik helyes volt), a kódszót érvényesnek fogadja el. Ha a számláló nem nulla, akkor a hibás bit sorszámát tartalmazza. Például, ha az 1., 2. és a 8. ellenőrző bit helytelen, akkor a megváltozott bit a 11., mert ez az egyetlen bit, amit



3.6. ábra. *Hamming-kód alkalmazása csoportos hibák javítására*

az 1., 2. és a 8. bitek ellenőriznek. A 3.6. ábra néhány 7 bites ASCII karaktert mutat 11 bites Hamming-kóddal kódolva. Ne felejtsük, hogy az adat a 3., 5., 6., 7., 9., 10. és a 11. pozíciókban található!

A Hamming-kódok csak egybites hibákat tudnak javítani. Van azonban egy trükk, amivel képessé tehetjük a Hamming-kódot csoportos hibák javítására. A kódszavak k hosszú sorozatát mátrixba rendezzük, minden sorba egy kódszót. Rendszerint az adatot kódszavanként továbbítjuk, a kódszó bitjeit pedig balról jobbra. Ahhoz hogy csoportos hibát tudjunk javítani, az adatot oszloponként kell továbbítani, a bal szélső oszloppal kezdve. Ha mind a k bitet elküldtük, a következő oszlop jön és így tovább. Amikor a keret megérkezik a vevőhöz, a mátrixot oszloponként újra felépítjük. Ha egy k hosszúságú csoporthiba következik be, az legfeljebb 1 bitet érint soronként, de mivel a Hamming-kód egy hibát kitud javítani kódszavanként, az egész blokkot helyre tudjuk állítani. Ez a módszer kr ellenőrző bitet használ ahhoz, hogy km méretű adatblokkokat egy legfeljebb k hosszúságú csoportos hiba ellen immúnissá tegyen.

3.2.2. Hibajelző kódok

A hibajavító kódokat csak ritkán alkalmazzák adatátvitelhez: például ha a csatorna egyirányú, így az adatok újraküldése nem kérhető. Sokkal gyakrabban a hibajelző kód és az adat újraküldés kombinációját részesítik előnyben, mert hatékonyabb. Egyszerű példaként vegyünk egy csatornát, amelyen a hibák izoláltak, és a hibaarány 10^{-6}. Legyen a blokkméret 1000 bit. Az 1000 bites blokkok hibajavításának biztosításához 10 ellenőrző bit szükséges; egy megabit adathoz 10 000 ellenőrző bit kell. Ahhoz, hogy csupán jelezzünk egy egybites hibát, egy paritásbit elegendő blokkonként: 1000 blokkonként egy többletblokkot (1001 bit) kell továbbítani. A hibajelzés és az adatújraküldés összes többletköltsége 2001 bit egy megabitnyi adatra, szemben a Hamming-kód 10 000 bitjével.

Ha a blokkot egyetlen paritásbittel egészítjük ki, és azt egy csoportos hiba erősen eltorzítja, a hiba jelzésének az esélye csupán 0,5, ami alig elfogadható. Az esély jelentősen növelhető, ha minden blokkot egy n bit szélességű, k bit hosszúságú mátrixnak tekintve küldünk el. Minden oszlophoz külön kiszámítjuk a paritásbitet, és utolsó sorként a mátrixhoz ragasztjuk őket. A mátrixot ezután soronként továbbítjuk. Amikor a blokk megérkezik, a vevő ellenőrzi az összes paritásbitet. Amennyiben bármelyik rossz, kéri a blokk újraküldését.

Ez a módszer egy n hosszúságú csoportos hibát tud jelezni, hiszen ez oszloponként csak 1 bitet változtat meg. Egy $n + 1$ hosszúságú csoport továbbra is észrevétlen marad, ha az első és az utolsó bit invertálódik, és az összes többi helyes. (A csoportos hiba nem jelenti azt, hogy az összes bit rossz, csak azt, hogy legalább az első és az utolsó az.) Ha a blokk erősen eltorzul egy hosszú vagy több rövidebb csoportos hiba miatt, annak az esélye, hogy az n oszlop bármelyikében véletlenül jó a paritásbit, 0,5. Így a valószínűsége annak, hogy egy blokkot elfogad a vevő, amikor nem kellene 2^{-n}.

Bár a fenti elgondolás esetenként megfelelő, a gyakorlatban egy másik módszert alkalmaznak széleskörűen: a **polinom-kódot (polynomial code),** más néven **ciklikus redundancia** vagy **CRC kódot (cyclic redundancy code).** A polinomkódok azon

alapulnak, hogy a bitsorozatokat polinomok reprezentációjának tekintjük, melyekben csupán a 0 és 1 együtthatók szerepelnek. Egy k bites keretet tekintsünk egy k tagú polinom együtthatóinak x^{k-1}-től x^0-ig. Az ilyen polinomot k-1-ed fokúnak nevezzük. A legnagyobb helyi értékű (bal szélső) bit az x^{k-1} együtthatója; a következő bit az x^{k-2}-é és így tovább. Például a 110001 6 bites, így egy 6 tagú polinomot reprezentál 1, 1, 0, 0, 0 és 1 együtthatókkal: $x^5 + x^4 + x^0$.

A polinom aritmetika modulo 2 végzendő az algebrai terek elmélete szerint. Nincs átvitel az összeadásnál és a kivonásnál, melyek a KIZÁRÓ VAGY művelettel azonosak. Például:

10011011	00110011	11110000	01010101
+ 11001010	+ 11001101	− 10100110	− 10101111
01010001	11111110	01010110	11111010

Az osztást ugyanúgy kell elvégezni, mint a binárist, kivéve, hogy a kivonás – mint fent – modulo 2 értendő. Az osztó „megvan" az osztandóban, ha az osztandó ugyanannyi bitet tartalmaz, mint az osztó.

Amikor a polinomkódot alkalmazzuk, az adónak és a vevőnek előre meg kell egyeznie egy **generátor polinomban (generator polynomial).** Jelöljük ezt $G(x)$-szel. A generátor legfelső és legalsó bitjének 1-nek kell lennie. Ahhoz, hogy egy $M(x)$ polinomnak megfelelő m bites keret ellenőrző összegét kiszámíthassuk, a keretnek hosszabbnak kell lennie, mint a generátor polinom. Az ötlet az, hogy úgy fűzzünk ellenőrző összeget a kerethez, hogy az így kapott keret által reprezentált polinom osztható legyen $G(x)$-szel. Amikor a vevő megkapja a keretet, megpróbálja elosztani $G(x)$-szel. Ha van maradék, akkor hiba volt az átvitel során.

Az ellenőrző összeg kiszámításának algoritmusa a következő:

1. Legyen r $G(x)$ foka. Fűzzünk r darab 0 bitet a keret alacsony helyi értékű végéhez, így az $m + r$ bitet fog tartalmazni és az $x^rM(x)$ polinomot fogja reprezentálni.

2. Osszuk el az $x^rM(x)$-hez tartozó bitsorozatot a $G(x)$-hez tartozó bitsorozattal modulo 2.

3. Vonjuk ki a maradékot (mely mindig r vagy kevesebb bitet tartalmaz) az $x^rM(x)$-hez tartozó bitsorozatból modulo 2-es kivonással. Az eredmény az ellenőrző összeggel ellátott, továbbítandó keret. Nevezzük ennek a polinomját $T(x)$-nek.

A 3.7. ábra illusztrálja a számítást egy 1101011011-et tartalmazó keretre és $G(x) = x^4 + x + 1$-re.

Tisztán látszik, hogy $T(x)$ osztható (modulo 2) $G(x)$-szel. Bármilyen osztási problémánál, ha az osztandót csökkentjük a maradékkal, az eredmény osztható lesz az osztóval. Például 10-es számrendszerben, ha elosztjuk 210 278-at 10 941-gyel, a maradék 2399. Kivonva 2399-et 210 278-ból, az eredmény (207 879) osztható 10 941-gyel.

Most pedig elemezzük a módszer erejét! Milyen hibákat fogunk tudni jelezni? Képzeljük el, hogy átviteli hiba történt, így a $T(x)$-nek megfelelő bitsorozat helyett

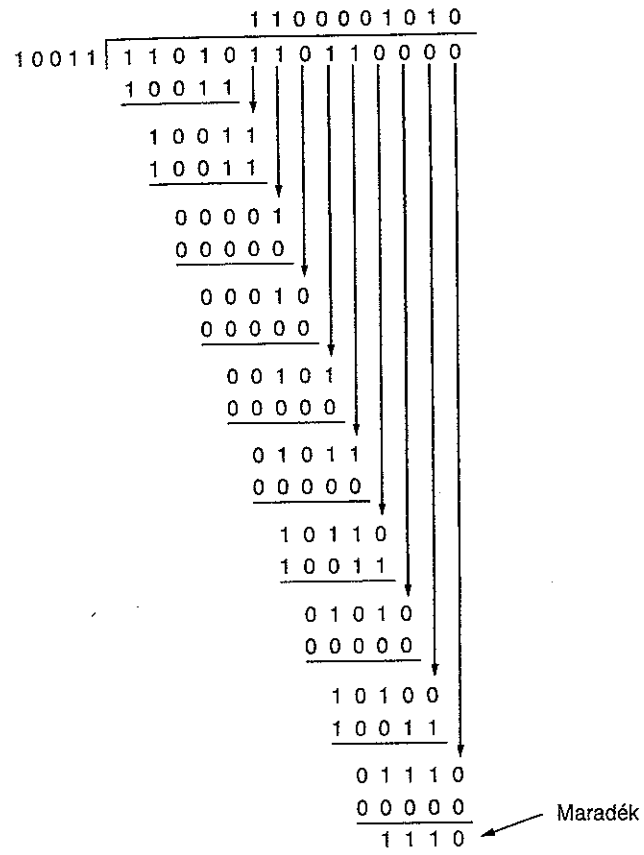
$T(x) + E(x)$-nek megfelelő érkezik. Minden 1-es bit $E(x)$-ben egy invertálódott bitnek felel meg. Ha k darab 1-es bit van $E(x)$-ben, k darab egybites hiba történt. Egy csoportos hiba egy kezdeti 1-gyel, 0-k és 1-ek keverékével, és egy záró 1-gyel jelképezhető. $E(x)$ többi bitje ebben az esetben 0.

Az ellenőrző összeggel ellátott keretet a vevő elosztja $G(x)$-szel; azaz kiszámítja $[T(x) + E(x)]/G(x)$-et. $T(x)/G(x)$ egyenlő 0-val, így a számítás eredménye csak az

Keret : 1 1 0 1 0 1 1 0 1 1

Generátor: 1 0 0 1 1

Az üzenet 4 darab 0 bit hozzáfűzése után: 1 1 0 1 0 1 1 0 1 1 0 0 0 0



Továbbított keret: 1 1 0 1 0 1 1 0 1 1 1 1 1 0

3.7. ábra. *A polinomkódot ellenőrző összeg kiszámítása*

$E(x)/G(x)$. Azok a hibák, melyek történetesen olyan polinomnak felelnek meg, amelyek $G(x)$ többszörösei, átcsúsznak az ellenőrzésen; minden más hibát felismer a vevő.

Ha egy egybites hiba történt, $E(x) = x^i$, ahol i a hibás bit sorszáma. Ha $G(x)$ kettő vagy több tagból áll, soha nem fogja osztani $E(x)$-et, így minden egybites hibát jelezni fogunk.

Ha két izolált egybites hiba történt, $E(x) = x^i + x^j$, ahol $i > j$. Másképpen ez az $E(x) = x^j(x^{i-j} + 1)$ alakba írható. Ha feltételezzük, hogy $G(x)$ nem osztható x-szel akkor, ahhoz, hogy minden kettős hibát jelezni tudjunk, $G(x)$-nek nem szabad osztania $x^k + 1$-et semmilyen az $i - j$ maximális értékénél (azaz, a maximális kerethossznál) kisebb k-ra. Egyszerű, alacsony fokszámú polinomok ismertek hosszú keretek védelmére. Például az $x^{15} + x^{14} + 1$ nem osztja $x^k + 1$-et semmilyen 32 768-nál kisebb k-ra.

Ha páratlan számú bit hibás, $E(x)$ páratlan számú tagot tartalmaz (pl. $x^5 + x^2 + 1$, de pl. az $x^2 + 1$ nem ilyen). Elég érdekes, hogy nincs olyan páratlan tagot tartalmazó polinom, amely osztható lenne $x + 1$-gyel a modulo 2-es rendszerben. $G(x)$-et úgy megválasztva, hogy osztható legyen $x + 1$-gyel, az összes páratlan számú invertált bitet tartalmazó hibát felismerhetjük.

Hogy belássuk azt, hogy nincs olyan páratlan számú tagot tartalmazó polinom, amely osztható $x + 1$-gyel, tételezzük fel, hogy $E(x)$ páratlan számú tagot tartalmaz, és osztható $x + 1$-gyel. Alakítsuk $E(x)$-et szorzattá: $E(x) = (x + 1)Q(x)$. Most értékeljük ki az $E(1) = (1 + 1)Q(1)$ egyenletet. Mivel $1 + 1 = 0$ (modulo 2), $E(1)$-nek is 0-nak kell lennie. Ha $E(x)$ páratlan számú tagból áll, x helyébe egyet helyettesítve mindig 1-et kapunk. Ezért nincs olyan páratlan számú tagból álló polinom, amely osztható $x + 1$-gyel.

Végül, a legfontosabb az, hogy egy r ellenőrző bittel ellátott polinomkód minden legfeljebb r hosszúságú csoportos hibát jelezni tud. Egy k hosszúságú csoportos hiba $x^i(x^{k-1} + \ldots + 1)$-gyel reprezentálható, ahol i azt mutatja, hogy a csoportos hiba a keret jobb szélétől milyen messze kezdődik. Ha $G(x)$ tartalmaz x^0 tagot, nem osztható x^i-nel, így ha a zárójeles kifejezés fokszáma kisebb, mint $G(x)$-é, a maradék sohasem lehet 0.

Ha a csoport hossza $r + 1$, a $G(x)$-szel való osztás maradéka akkor, és csak akkor lehet 0, ha a csoportos hiba $G(x)$-nek felel meg. A csoportos hiba definíciója miatt az első és az utolsó bit mindenképpen 1, így az, hogy a hiba megfelel-e $G(x)$-nek, az $r - 1$ közbenső biten múlik. Ha minden kombinációt egyenlően valószínűnek veszünk, akkor annak a valószínűsége, hogy egy ilyen hibás keretet a vevő elfogad: $1/2^{r-1}$.

Az is megmutatható, hogy amikor egy $r + 1$ bitnél hosszabb vagy több rövidebb csoportos hiba lép fel akkor, annak a valószínűsége, hogy egy hibás keret észrevétlen marad $1/2^r$, feltételezve, hogy minden bitminta egyformán valószínű.

Három polinom vált nemzetközi szabvánnyá:

$$\text{CRC–12} \quad = x^{12} + x^{11} + x^3 + x^2 + x^1 + 1$$

$$\text{CRC–16} \quad = x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC–CCITT} = x^{16} + x^{12} + x^5 + 1$$

Mindhárom osztható $x + 1$-gyel. A CRC–12-t akkor alkalmazzák, ha a karakterhossz 6. A másik kettő 8 bites karakterekhez használatos. Egy 16 bites ellenőrző összeg, mint például a CRC–16 vagy a CRC–CCITT felismer minden egybites és kétbites (dupla) hibát, minden páratlan számú hibás bitet tartalmazó hibát, valamint minden 16 vagy kevesebb bitnyi csoportos hibát, a 17 bites csoportos hibák 99,997 százalékát és a 18 vagy több bitesek 99,998 százalékát.

Bár az ellenőrző összeg kiszámításához szükséges algoritmus bonyolultnak tűnhet, Peterson és Brown (1961) megmutatta, hogy szerkeszthető egy egyszerű, léptető regiszteres áramkör az ellenőrző összeg hardverben történő kiszámítására és ellenőrzésére. A gyakorlatban majdnem mindig használják ezt a hardvert.

Évtizedek óta azt feltételezzük, hogy az ellenőrző összeggel ellátandó keretek véletlenszerű biteket tartalmaznak. Az ellenőrzőösszeg-képző algoritmusok minden analízise erre a feltételezésre alapult. A valóságos adatok vizsgálata egyre gyakrabban azt mutatja, hogy ez a feltételezés meglehetősen rossz. Következésképp bizonyos körülmények között az észrevétlen hibák sokkal gyakoribbak, mint ahogyan azt korábban gondoltuk (Partridge és mások, 1995).

3.3. Elemi adatkapcsolati protokollok

A protokollok témakörébe való bevezetésként, három, egyre növekvő bonyolultságú protokollt fogunk megnézni. Az érdeklődő olvasók számára ezekhez és további protokollokhoz egy szimulátor áll rendelkezésre a WWW-n (lásd a bevezetőt). Mielőtt megnéznénk a protokollokat, hasznos nyilvánvalóvá tenni néhány, a kommunikáció modelljét megalapozó feltételezést. Először is feltételezzük, hogy a fizikai réteg, az adatkapcsolati réteg és a hálózati réteg független folyamatok, melyek üzenetek oda-vissza küldözgetésével kommunikálnak egymással. Néhány esetben a fizikai és az adatkapcsolati réteg folyamatai egy speciális hálózati B/K chip processzorán futnak, míg a hálózati réteg a fő CPU-n. Persze más megvalósítások is lehetségesek (pl. három folyamat egyetlen B/K chipen belül; a fizikai és az adatkapcsolati réteg a hálózati réteg folyamata által hívott eljárások és így tovább). Bármelyik esetben a három réteg különálló folyamatnak tekintése fogalmilag világosabbá teszi a tárgyalást, és a rétegek függetlenségének hangsúlyozását is szolgálja.

Egy másik kulcsfontosságú feltételezés az, hogy az A gép megbízható, összeköttetés alapú szolgálat alkalmazásával akar a B gépnek egy hosszú adatfolyamot küldeni. Később azt az esetet is átgondoljuk, amikor B is akar adatot küldeni A-nak egyidejűleg. A-ról feltételezzük, hogy végtelen utánpótlása van elküldendő adatokból és sohasem kell várakoznia az adatok előállítására. Amikor A adatkapcsolati rétege adatot kér, a hálózati réteg mindig azonnal tudja teljesíteni a kérést. (Ezt a megkötést később szintén elvetjük.)

Ami az adatkapcsolati réteget illeti, a hálózati réteg által az interfészen keresztül hozzá eljuttatott csomag tiszta adat, aminek minden bitjét továbbítani kell a célgép hálózati rétegéhez. Az, hogy a hálózati réteg a csomag egy részét fejrésznek értelmezi, nem tartozik az adatkapcsolati rétegre.

Amikor az adatkapcsolati réteg megkap egy csomagot, egy keretbe burkolja, hozzáadva az adatkapcsolati fejrészt és farokrészt (lásd 1.11. ábra). Így egy keret egy be-

ágyazott csomagból és vezérlő (fejrész) információból áll. A keretet ezután elküldjük a másik adatkapcsolati rétegnek. Feltételezzük, hogy léteznek alkalmas könyvtári eljárások egy keret elküldésére (*to_physical_layer*) és vételére (*from_phisycal_layer*). A továbbítást végző hardver kiszámítja és ellenőrzi az ellenőrző összeget, így az adatkapcsolati réteg szoftverének nem kell ezzel foglalkoznia. Alkalmazható például az ebben a fejezetben korábban tárgyalt polinom algoritmus.

Kezdetben a vevőnek nincs mit tennie. Csak üldögél, várva, hogy valami történjen. A fejezet példa protokolljaiban a *wait_for_event(&event)* eljáráshívással jelezzük, hogy az adatkapcsolati réteg vár valamire. Ez az eljárás csak akkor tér vissza, ha valami történt (pl. egy keret érkezett). A visszatéréskor az *event* változó mondja meg, hogy mi volt az esemény. A lehetséges események halmaza a különféle ismertetett protokolloknál különböző, és mindegyik protokollhoz külön fogjuk meghatározni. Megjegyezzük, hogy egy valóságosabb helyzetben az adatkapcsolati réteg nem ücsörög egy végtelen ciklusban eseményre várva, ahogyan javasoltuk, hanem egy megszakítást kezel, aminek beérkezésekor félbeszakítja addigi tevékenységét, és a beérkező keret feldolgozását kezdi el. Ennek ellenére az egyszerűség kedvéért nem veszünk tudomást az adatkapcsolati rétegben folyó párhuzamos tevékenységekről, és feltételezzük, hogy a szoftver dolga kizárólag a mi egyetlen csatornánk kezelése.

Amikor egy keret érkezik a vevőhöz, a hardver kiszámítja az ellenőrző összeget. Ha az helytelen (azaz átviteli hiba történt), az adatkapcsolati réteget tájékoztatja erről (*event = cksum_err*). Ha a keret sértetlenül érkezik meg, az adatkapcsolati réteg erről szintén tájékoztatást kap (*event = frame_arrival*), így az megkaphatja a keretet a fizikai rétegtől a *from_physical_layer* eljárás használatával. Amint a vevő adatkapcsolati rétege megkapta a sértetlen keretet, ellenőrzi a vezérlőinformációkat a fejrészben, és ha minden rendben van, a csomag részt továbbadja a hálózati rétegnek. A keret fejrésze semmilyen körülmények között sem kerül a hálózati réteghez.

Jó oka van annak, hogy a hálózati rétegnek sohasem szabad átadni a keret fejrészének semmilyen részét: a hálózati és az adatkapcsolati protokollt teljesen szét kell választani. Amíg a hálózati réteg semmit sem tud az adatkapcsolati protokollról vagy a keretformátumról, ezek anélkül változhatnak meg, hogy bármilyen változtatást kellene tenni a hálózati réteg szoftverében. A hálózati réteg és az adatkapcsolati réteg között merev interfészt biztosítva, nagyon leegyszerűsödik a szoftver tervezése, mert így a különböző rétegekben levő kommunikációs protokollok egymástól függetlenül fejleszthetők.

A 3.8. ábra néhány deklarációt mutat be (C nyelven), amelyek több, később ismertetendő protokollban is megtalálhatók. Öt adatstruktúrát definiáltunk: *boolean*, *seq_nr*, *packet*, *frame_kind* és *frame*. A *boolean* egy felsorolt típus, és a *true* és a *false* értékeket veheti fel. A *seq_nr* egy kis egész szám, melyet a keretek megszámozására használunk, hogy meg tudjuk őket különböztetni. Ezek a sorszámok 0 és *MAX_SEQ* közötti értéket vehetnek fel, *MAX_SEQ*-t is beleértve, mely minden egyes protokollban definiálva van, amelyiknek szüksége van rá. A *packet* az az információegység, amit az azonos gépen levő hálózati és adatkapcsolati réteg vagy a kommunikáló hálózati réteg párok cserélnek ki egymással. A mi modellünkben ez mindig *MAX_PKT* számú bájtot tartalmaz, de a valóságnak jobban megfelelne, ha változó hosszúságú lenne.

A *frame* négy mezőből áll: *kind*, *seq*, *ack* és *info*, melyekből az első három vezérlő információt tartalmaz, az utolsó pedig a valódi továbbítandó adatot. Ezeket a vezérlő mezőket együttesen a **keret fejrészének (frame header)** nevezzük. A *kind* mező azt mondja meg, hogy van-e adat a keretben, néhány protokoll ugyanis különbséget tesz azok között a keretek között, amelyek kizárólag vezérlő információt tartalmaznak és amelyek adatot is. A *seq* és az *ack* mezőket sorszámozáshoz, illetve nyugtákhoz használjuk; később részletesebben ismertetjük a használatukat. A keret *info* mezője egyetlen csomagot tartalmaz; egy vezérlő keret *info* mezőjét nem használjuk. Egy a valóságnak jobban megfelelő megvalósításban változó hosszúságú *info* mezőt kellene használnunk, és a vezérlő kereteknél ezt a mezőt el is kellene hagynunk.

Fontos, hogy tisztában legyünk a csomag és a keret egymáshoz való viszonyával. A hálózati réteg úgy építi fel a csomagot, hogy vesz egy üzenetet a szállítási rétegtől és hozzáteszi a hálózati réteg fejrészét. Ezt a csomagot átadja az adatkapcsolati rétegnek, hogy az beletegye egy kimenő keret *info* mezőjébe. Amikor a keret megérkezik a célhoz, az adatkapcsolati réteg kiveszi a csomagot a keretből és átadja a hálózati rétegnek. Ilyen módon a hálózati réteg úgy működhet, mintha a gépek közvetlenül csomagokat tudnának váltani.

Számos eljárást is felsoroltunk a 3.8. ábrán. Ezek könyvtári rutinok, melyek részletei megvalósításfüggőek, és belső működésükkel itt nem foglalkozunk a továbbiakban. A *wait_for_event* eljárás egy végtelen ciklusban ücsörög, várva, hogy valami történjen, mint ahogyan korábban már említettük. A *to_network_layer* és a *from_network_layer* eljárásokat az adatkapcsolati réteg arra használja, hogy csomagokat adjon át a hálózati rétegnek, illetve vegyen át a hálózati rétegtől. Vegyük észre, hogy a *from_physical_layer* és a *to_physical_layer* eljárásokat keretek átadására használjuk az adatkapcsolati és a fizikai réteg között, míg a *to_network_layer* és a *from_network_layer* eljárásokat csomagok átadására az adatkapcsolati réteg és a hálózati réteg között. Más szóval, a *to_network_layer* és a *from_network_layer* a 2. és 3. rétegek közötti interfésszel foglalkozik, míg a *from_physical_layer* és a *to_physical_layer* az 1. és 2. réteg közöttivel.

A legtöbb protokollban megbízhatatlan csatornát tételezünk fel, amely alkalomadtán egész kereteket elveszíthet. Ahhoz, hogy ilyen csapásokból képes legyen felépülni, a küldő adatkapcsolati rétegnek minden keret elküldésekor el kell indítania egy belső időzítőt vagy órát. Ha bizonyos, előre meghatározott időn belül nem érkezik válasz, az óra lejár és az adatkapcsolati réteg egy megszakítás jelzést kap.

A protokolljainkban úgy oldjuk ezt meg, hogy a *wait_for_event* eljárás visszatérhet *timeout* eseménnyel. A *start_timer* és *stop_timer* eljárásokat az időzítő be-, illetve kikapcsolására használjuk. Az időtúllépés csak akkor következhet be, ha az időzítő megy. A *start_timer* meghívható akkor is, ha az időzítő megy; egy ilyen hívás egyszerűen lenullázza az órát, hogy a következő időtúllépés egy teljes időzítési intervallum után következzen be (hacsak közben az időzítőt nem nullázzuk le újra, vagy nem kapcsoljuk ki).

A *start_ack_timer* és a *stop_ack_timer* eljárásokat a segéd időzítő vezérlésére használjuk. Ez az időzítő állít elő nyugtákat bizonyos körülmények között.

```
#define MAX_PKT 1024                    /* megadja a keretek hosszát bájtban */

typedef enum {true, false} boolean;     /* logikai típus */
typedef unsigned int seq_nr;            /* sorszámok vagy nyugta számok */
typedef struct {unsigned char data[MAX_PKT];} packet; /* csomag definíció */
typedef enum {data, ack, nak} frame_kind;   /* keret fajta definíció */

typedef struct {                        /* ebben a rétegben kereteket továbbítunk */
  frame_kind kind;                      /* milyen fajta keret? */
  seq_nr seq;                           /* sorszám */
  seq_nr ack;                           /* nyugta száma */
  packet info;                          /* a hálózati rétegbeli csomag */
} frame;

/* Vár egy esemény bekövetkezésére; visszaadja a típusát az event-ben. */
void wait_for_event(event_type *event);

/* Lehoz egy csomagot a hálózati rétegtől a csatornán való továbbításra. */
void from_network_layer(packet *p);

/* Az érkező keretből átadja az információt a hálózati rétegnek.*/
void to_network_layer(packet *p);

/* Átveszi az érkező keretet a fizikai rétegtől és r-be másolja. */
void from_physical_layer(frame *r);

/* Átadja a keretet a fizikai rétegnek továbbításra. */
void to_physical_layer(frame *s);

/* Elindítja az órát és engedélyezi a timeout eseményt. */
void start_timer(seq_nr k);

/* Leállítja az órát és letiltja a timeout eseményt. */
void stop_timer(seq_nr k);

/* Elindítja a segéd időzítőt és engedélyezi az ack_timeout eseményt. */
void start_ack_timer(void);

/* Leállítja a segéd időzítőt és letiltja az ack_timeout eseményt. */
void stop_ack_timer(void);

/* Engedélyezi a hálózati rétegnek, hogy network_layer_ready eseményt okozzon. */
void enable_network_layer(void);

/* Megtiltja a hálózati rétegnek, hogy network_layer_ready eseményt okozzon. */
void disable_network_layer(void);

/* Az inc makro soron belül (in-line) lesz kifejtve: körkörösen növeli k-t. */
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

3.8. ábra. *Néhány definíció, amely a későbbi protokollokhoz szükséges. Ezek a definíciók a protocol.h fájlban találhatók*

Az *enable_network_layer* és a *disable_network_layer* eljárásokat azok a kifinomultabb protokollok használják, melyekben már nem tételezzük fel, hogy a hálózati rétegnek mindig van küldeni való csomagja. Amikor az adatkapcsolati réteg engedélyezi a hálózati réteget, a hálózati réteg megszakítást küldhet, ha van elküldeni való csomagja. Ezt az *event = network_layer_ready*-vel jelezzük. Amikor a hálózati réteg le van tiltva, nem okozhat ilyen eseményt. Ügyelve arra, hogy mikor engedélyezi és tiltja le a hálózati rétegét, az adatkapcsolati réteg megakadályozhatja, hogy a hálózati réteg elárassza csomagokkal, amelyek tárolására nincs pufferterülete.

A keretsorszámok mindig a 0–*MAX_SEQ* (zárt) intervallumon belül vannak, ahol *MAX_SEQ* a különböző protokollokhoz különböző. Gyakran szükséges, hogy egy sorszámot 1-gyel továbbléptessünk körkörösen (azaz *MAX_SEQ*-t 0 követi). Az *inc* makró ezt a léptetést végzi. Azért definiáltuk makróként, mert a kritikus végrehajtási útvonalon belül soron belül kifejtve (in-line) használjuk. Ahogyan a könyv későbbi részében látni fogjuk, az a tényező, mely korlátozza a hálózat teljesítőképességét, gyakran a protokoll végrehajtása, így ha az olyan egyszerű műveleteket, mint ez, makróként definiáljuk, ez nem befolyásolja a kód olvashatóságát, de javítja a teljesítőképességét. Hasonlóan, mivel a *MAX_SEQ* különböző protokollok esetében különböző értéket vehet fel, makróként definiálva lehetséges egyetlen bináris állományba tenni az összes protokollt anélkül, hogy zavarnák egymást. Ez a lehetőség a szimulátornak hasznos.

A 3.8. ábrán levő deklarációk mindegyike, a következőkben ismertetendő protokollban megtalálható. Helytakarékosságból és a kényelmesebb hivatkozás miatt, kiemeltük őket és együtt soroltuk fel, de fogalmilag magukkal a protokollokkal összevonva kell érteni őket. A C nyelvben ezt az összevonást a definíciók speciális fejrész fájlba (ebben az esetben a *protocol.h*) helyezésével, és a C előfeldolgozó #include lehetőségének használatával érjük el, így a definíciók a fordításkor a protokoll fájlokban megjelennek.

3.3.1. Korlátozás nélküli szimplex protokoll

Bevezető példaként egy olyan protokollt veszünk, amelyik annyira egyszerű, amennyire csak lehet. Az adatokat csak egy irányba továbbítjuk. Mind az adó, mind a vevő hálózati rétegei mindig készen állnak. A feldolgozási időtől eltekinthetünk. Végtelen pufferterület áll rendelkezésre. És a legjobb: az adatkapcsolati rétegek közötti kommunikációs csatorna sohasem rontja vagy veszíti el a kereteket. Ez a meglehetősen valószerűtlen protokoll, melyet „utópiá"-nak fogunk becézni, a 3.9. ábrán látható.

A protokoll két különálló eljárásból épül fel: egy küldőből és egy vevőből. A küldő a forrásgép adatkapcsolati rétegében fut, a vevő pedig a célgépében. Itt semmilyen sorszámozást vagy nyugtázást nem használunk, így a *MAX_SEQ*-ra nincs szükség. Az egyetlen lehetséges eseményfajta a *frame_arrival* (azaz egy sértetlen keret érkezése).

A küldő egy végtelen while ciklus, amely olyan gyorsan pumpálja kifelé a vonalra az adatokat, ahogy csak tudja. A ciklusmag három teendőből áll: szerezni egy csomagot (a mindig szolgálatkész) hálózati rétegtől, összerakni egy keretet az *s* változó segítségével, és útjára indítani a keretet. Ez a protokoll csak a keret *info* mezőjét használja, mivel a többi mező a hibajavításhoz és a forgalomszabályozáshoz kell, és itt nincsenek hibák vagy forgalomszabályozási megkötések.

```
/* Az 1. protokoll (utópia) csak egyirányú adatátvitelről gondoskodik: az adótól a vevő
   felé. A kommunikációs csatornát hibamentesnek feltételezzük. A vevőről azt
   tételezzük fel, hogy képes a bemenő adatokat végtelen gyorsan feldolgozni.
   Következésképp az adó egy ciklusban pumpálja ki az adatokat a vonalra olyan
   gyorsan, ahogyan csak tudja. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender1(void)
{
  frame s;                          /* puffer a kimenő keretnek */
  packet buffer;                    /* puffer a kimenő csomagnak */

  while (true) {
    from_network_layer(&buffer);    /* szerezzünk valami küldenivalót */
    s.info = buffer;                /* másoljuk s-be a továbbításhoz */
    to_physical_layer(&s);          /* eresszük útnak */
  }                                 /* Holnap és holnap és holnap: tipegve
                                     vánszorog létünk a kimért idő
                                     végső szótagjáig ...
                                     - Macbeth V. felv., 5. szín
                                     / ford. Szabó Lőrinc /   */
}
void receiver2(void)
{
  frame r;
  event_type event;                 /* a wait_for_event tölti ki, de itt nem használjuk */

  while (true) {
    wait_for_event(&event);         /*az egyetlen lehetőség a frame_arrival */
    from_physical_layer(&r);        /* szerezzük meg a bejövő keretet */
    to_network_layer(&r.info);      /* adjuk tovább az adatot a hálózati rétegnek */
  }
}
```

3.9. ábra. *Korlátozás nélküli szimplex protokoll*

A vevő ugyanilyen egyszerű. Kezdetben vár, hogy valami történjen. Az egyetlen lehetőség egy sértetlen keret érkezése. Végül is megérkezik a keret, és a *wait_for_event* eljárás visszatér az *event*-et *frame_arrival*-re állítva (melyet egyébként nem veszünk figyelembe). A *from_physical_layer* meghívása eltávolítja a keretet a hardver pufferéből, és beleteszi az *r* változóba. Végül az adatrészt továbbadjuk a hálózati rétegnek, és az adatkapcsolati réteg dolga végeztével várakozni kezd a következő keretre: felfüggeszti magát, amíg a keret meg nem érkezik.

3.3.2. Szimplex megáll-és-vár protokoll

Most elvetjük az 1. protokollban alkalmazott legkevésbé valósághű megkötést: a vevő adatkapcsolati rétegének azt a képességét, hogy végtelenül gyorsan fel tudja dolgozni a bejövő adatokat (vagy, ami ezzel azonos, azt, hogy a vevő adatkapcsolati rétegében végtelen méretű pufferterület van, amelyben a bejövő keretek tárolódnak, amíg sorra nem kerülnek a feldolgozásban). A kommunikációs csatornát azonban továbbra is hibamentesnek feltételezzük, és az adatforgalom is egyirányú.

A fő probléma, amivel itt foglalkoznunk kell az, hogy megakadályozzuk a küldő állomást abban, hogy gyorsabban adjon adatokat, mint ahogy a vevő fel tudná őket dolgozni. Lényegében: ha a vevőnek Δt időre van szüksége a *from_physical_layer* és a *to_network_layer* végrehajtására, a küldő állomásnak átlagosan kevesebb, mint egy keretet szabad küldenie Δt idő alatt. Továbbá, ha feltételezzük, hogy nincs automatikus pufferelés és sorba állítás a vevő hardverén belül, a küldőnek nem szabad a következő keretet leadnia addig, amíg az előzőt a *from_physical_layer* hívás segítségével a vevő ki nem vette a pufferből, különben az új keret felülírná a régit.

Bizonyos speciális körülmények között (pl. szinkron átvitel, és a vevő adatkapcsolati rétege kizárólag egyetlen bejövő vonallal foglalkozik), lehetséges lenne, hogy a küldő állomás egyszerűen egy késleltetést iktasson be az 1. protokollba, hogy eléggé lelassítsa magát ahhoz, hogy ne tudja elárasztani a vevőt. Sokkal gyakoribb azonban az, hogy az adatkapcsolati rétegnek több vonalra kell figyelnie, és a keretek beérkezése és feldolgozása közötti idő jelentősen változhat. Ha a hálózattervezők ki tudják számítani a vevő viselkedését a legrosszabb esetre, akkor be tudják programozni a küldő állomást úgy, hogy olyan lassan adjon, hogy még ha minden keret maximális késleltetést is szenved, akkor se legyen torlódás. A probléma ezzel a megközelítéssel az, hogy túl konzervatív: olyan sávszélesség-kihasználtsághoz vezet, ami messze az optimális alatt van, kivéve, ha a legjobb és a legrosszabb eset szinte megegyezik (azaz az adatkapcsolati réteg reakcióidejének szórása kicsi).

Egy sokkal általánosabb megoldás erre a problémára az, hogy a vevő visszacsatolást biztosít a küldő állomás felé. Miután a vevő átadta a csomagot a hálózati rétegének, visszaküld egy kis álkeretet, amely valójában engedély a küldő állomásnak arra, hogy továbbítsa a következő keretet. Miután az állomás egy keretet elküldött, a protokoll szerint várakoznia kell addig, amíg a kis ál- (azaz nyugta-) keret meg nem érkezik.

Azokat a protokollokat, melyekben a küldő egy keret elküldése után nyugtára vár, mielőtt továbbmenne, **megáll-és-vár (stop-and-wait)** protokollnak nevezik. A 3.10. ábrán egy példa látható a szimplex megáll-és-vár protokollra.

Úgy, mint az 1. protokollban, a küldő állomás itt is azzal kezdi, hogy lekér egy csomagot a hálózati rétegtől, ennek segítségével összeállít egy keretet, és útjára engedi. Csakhogy most – nem úgy, mint az 1. protokollban – a küldőnek várakoznia kell, amíg egy nyugtakeret meg nem érkezik, mielőtt újabb ciklusba kezdene, és lekérné a következő csomagot a hálózati rétegtől. A küldő adatkapcsolati rétegnek meg sem kell néznie az érkező keretet: úgyis csak egy lehetőség van...

Az egyetlen különbség a *receiver1* és a *receiver2* között az, hogy miután a csomagot kézbesítette a hálózati rétegnek, a *receiver2* egy nyugtakeretet küld vissza a küldő

```
/* A 2. protokoll (megáll-és-vár) szintén egyirányú adatátvitelről gondoskodik: a
   küldőtől a vevő felé. A kommunikációs csatornát megint hibamentesnek
   feltételezzük, mint az 1. protokollban. Ebben az esetben azonban a vevőnek csak
   véges puffer kapacitása és véges feldolgozási sebessége van, így a protokollnak
   explicit módon meg kell akadályoznia a küldő állomást abban, hogy gyorsabban
   továbbítsa az adatokat a vevő felé, mint ahogy az kezelni tudná. */

typedef enum {frame_arrival} event_type;
#include "protocol.h"

void sender2(void)
{
  frame s;                          /* puffer a kimenő keretnek */
  packet buffer;                    /* puffer a kimenő csomagnak */
  event_type event;                 /* frame_arrival az egyetlen lehetőség */

  while (true) {
    from_network_layer(&buffer);    /* szerezzünk valami küldenivalót */
    s.info = buffer;                /* másoljuk s-be a továbbításhoz */
    to_physical_layer(&s);          /* szia kicsi keret */
    wait_for_event(&event);         /* ne folytassuk, amíg nem halljuk, hogy
                                       előre! */
  }
}
void receiver2(void)
{
  frame r, s;                       /* pufferek kereteknek */
  event_type event;                 /* frame_arrival az egyetlen lehetőség */

  while (true) {
    wait_for_event(&event);         /*az egyetlen lehetőség a frame_arrival */
    from_physical_layer(&r);        /* szerezzük meg a bejövő keretet */
    to_network_layer(&r.info);      /* adjuk tovább az adatot a hálózati
                                       rétegnek */
    to_physical_layer(&s);          /* küldjünk egy ál-keretet a küldő
                                       állomás felébresztésére */
  }
}
```

3.10. ábra. *Szimplex megáll-és-vár protokoll*

állomásnak a várakozó hurokba való újbóli belépés előtt. Mivel a küldőnek csak a keret visszaérkezése fontos, az nem, hogy mit tartalmaz, a vevőnek nem szükséges semmilyen különleges információt beletenni.

Bár ebben a példában az adatáramlás egyirányú: csak a küldőtől a vevő felé folyik, keretek mindkét irányban utaznak. Következésképp, a két adatkapcsolati réteg közötti kommunikációs csatornának alkalmasnak kell lennie kétirányú információátvitelre. A

protokollból következik az áramlás irányának szigorú váltakozása: először a küldő állomás küld egy keretet, azután a vevő küld egy keretet, azután a küldő küld még egy keretet, azután a vevő még egyet és így tovább. Ehhez egy félduplex fizikai csatorna lenne megfelelő.

3.3.3. Szimplex protokoll zajos csatornához

Most vegyük figyelembe, hogy a kommunikációs csatorna – mint általában – hibázhat. A keretek megsérülhetnek, vagy teljesen elveszhetnek. Azt azért feltételezzük, hogy ha egy keret megsérül az átvitel során, a vevő hardvere ezt felismeri, amikor kiszámítja az ellenőrző összeget. Ha a keret úgy sérül meg, hogy az ellenőrző összeg ennek ellenére helyes – ami rendkívül ritkán fordul elő –, ez a protokoll (és az összes többi is) hibázhat (azaz egy hibás keretet kézbesíthetnek a hálózati rétegnek).

Első pillantásra úgy tűnhet, hogy a 2. protokoll egy kis változtatással jó lenne: be kellene építeni egy időzítőt. A küldő állomás küldhetne egy keretet, de a vevő csak akkor küldene nyugtakeretet, ha az adat helyesen érkezett meg. Ha sérült keret érkezik a vevőhöz, el kellene dobni. Egy idő után a küldőnek lejárna az időzítője, és újra elküldené a keretet. Ezt a folyamatot addig ismételnénk, míg végül a keret sértetlenül megérkezik.

A fenti sémának van egy végzetesen gyenge pontja. El kellene gondolkodni, és mielőtt továbbmegyünk, rá kellene jönni, hogy mi lehet a gond!

Ahhoz, hogy lássuk, hogy mi a baj, emlékezzünk rá, hogy az adatkapcsolati réteg folyamatainak a kötelessége, hogy hibamentes, átlátszó kommunikációt biztosítsanak a hálózati rétegek folyamatai között. Az *A* gép hálózati rétege egy csomagsorozatot ad az adatkapcsolati rétegnek, amelynek biztosítania kell, hogy megegyező csomagsorozatot kapjon a hálózati réteg a *B* gépen az adatkapcsolati rétegtől. Különös tekintettel arra, hogy a *B* gép hálózati rétegének nincs módja tudomást szerezni arról, hogy egy csomag elveszett vagy megkettőződött, az adatkapcsolati rétegnek kell garantálnia, hogy az átviteli hibák semmilyen kombinációja – tekintet nélkül arra, hogy milyen ritkán fordulnak elő – sem okozhassa, hogy egy keret duplán érkezzen meg a hálózati réteghez.

Gondoljuk át a következő eseménysort:

1. Az *A* hálózati rétege átadja az 1. csomagot az adatkapcsolati rétegnek. A csomag rendben megérkezik *B*-hez, és átadódik *B* hálózati rétegének. *B* visszaküld egy nyugtakeretet *A*-nak.

2. A nyugtakeret teljesen elveszik. Egyáltalán nem érkezik meg soha. Az élet sokkal egyszerűbb lenne, ha a csatorna csak az adatkereteket tenné tönkre és veszítené el, a vezérlőkereteket nem, de igaz, ami igaz, a csatorna nem nagyon tesz különbséget.

3. *A* adatkapcsolati rétegének végül lejár az időzítője. Mivel nem érkezett nyugta, azt feltételezi (hibásan), hogy az adatkerete elveszett vagy megsérült, így újra elküldi az 1. csomagot tartalmazó keretet.

4. A megkettőzött keret szintén tökéletesen megérkezik *B* adatkapcsolati rétegéhez és az minden rossz szándék nélkül átadja az ottani hálózati rétegnek. Ha *A* egy állományt küld *B*-nek, az állomány egy része megkettőződik (azaz *B* által az állományról készített másolat hibás lesz, és a hibát nem vesszük észre). Egyszóval, a protokoll hibázik.

Világos, hogy arra van szükség, hogy valahogyan képessé tegyük a vevőt arra, hogy meg tudja különböztetni az először látott kereteket az újraküldöttektől. Ennek kézenfekvő módja az, hogy az adó egy sorszámot tesz minden elküldött keret fejlécébe. Ekkor a vevő ellenőrizheti minden érkező keret sorszámát, hogy megállapítsa, hogy egy új keret érkezett-e, vagy egy megkettőzött, amit el kell dobni.

Mivel a kisméretű keretfejrész a kívánatos, adódik a kérdés, hogy minimálisan hány bit szükséges a sorszám tárolásához. A protokollban az egyetlen nem egyértelmű helyzet az *m*. és a közvetlenül azt követő, *m* + 1. keret között áll fenn. Ha az *m*. keret elveszett vagy megsérült, a vevő nem fogja nyugtázni, így a küldő tovább próbálkozik az elküldésével. Ha egyszer hibátlanul megérkezik, a vevő visszaküld egy nyugtát a küldő állomásnak. Ez az a pont, ahol a potenciális hiba fellép. Attól függően, hogy a nyugtakeret rendben visszajut a küldő állomáshoz vagy sem, az az *m*. vagy az *m* + 1. keretet próbálja meg elküldeni.

Az az esemény, mely kiváltja az *m* + 2. keret elküldését, az *m* + 1.-re vonatkozó nyugta megérkezése. Ez azonban azt is magában foglalja, hogy az *m*. is rendben megérkezett, továbbá az *m*. keret nyugtája is megérkezett a küldő állomáshoz (másképp a küldő nem is fogott volna hozzá az *m* + 1. adásához, és pláne békén hagyta volna az *m* + 2.-ot). Következésképp az egyetlen bizonytalanság egy keret és az azt közvetlenül megelőző vagy követő keret között van, nem pedig a keretet megelőző és a keretet követő között.

A fentiek miatt egy 1 bites (0 vagy 1) sorszám elegendő. Minden pillanatban a vevő pontosan tudja, hogy milyen sorszámot vár. Minden rossz sorszámot tartalmazó keretet – mint duplikátumot – elutasít. Amikor egy helyes sorszámot tartalmazó keret érkezik, azt elfogadja, és átadja a hálózati rétegnek, majd a várt sorszámot modulo 2 lépteti (azaz a 0-ból 1 lesz, az 1-ből pedig 0).

A 3.11. ábrán egy példa látható ilyen fajta protokollra. Azokat a protokollokat, amelyekben a küldő állomás pozitív nyugtára vár, mielőtt továbblépne a következő adategységre, gyakran **PAR-nak (Positive Acknowledgement with Retransmission – pozitív nyugtázás újraküldéssel)** vagy **ARQ-nak (Automatic Repeat reQuest – automatikus ismétléskérés)** nevezik. Hasonlóan a 2. protokollhoz, ez is csak egy irányba továbbítja az adatokat. Bár tudja kezelni az elveszett kereteket (az időzítő segítségével), az időzítési intervallumnak olyan hosszúnak kell lennie, hogy elkerüljük az idő előtti időtúllépéseket. Ha a küldő állomásnak túl korán jár le az időzítője (azaz amíg a nyugta még úton van), duplikátumot fog küldeni.

Amikor az előző nyugta végül megérkezik, a küldő állomás tévesen azt gondolhatja, hogy az éppen elküldött keretet nyugtázta a vevő, és nem ismeri fel, hogy lehet egy másik nyugtakeret is valahol a „csőben". Ha a következő elküldött keret teljesen elveszik, de a plusznyugta hibátlanul megérkezik, a küldő nem fogja megpróbálni újraküldeni az elveszett keretet, és a protokoll hibázni fog. Ahhoz, hogy az ilyenfajta hibát


```
/* A 3. protokoll (PAR) egyirányú adatfolyamot biztosít megbízhatatlan csatornán. */

#define MAX_SEQ 1                       /* 1-nek kell lennie a 3. protokollhoz */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void sender3(void)
{
  seq_nr next_frame_to_send;            /* a következő kimenő keret sorszáma */
  frame s;                              /* puffer egy keretnek */
  packet buffer;                        /* puffer a kimenő csomagnak */
  event_type event;

  next_frame_to_send = 0;               /* inicializáljuk a kimenő sorszámot */
  from_network_layer(&buffer);          /* kérjük le az első csomagot */
  while (true) {
    s.info = buffer;                    /* másoljuk s-be a továbbításhoz */
    s.seq = next_frame_to_send;         /* tegyük bele a keretbe a sorszámot */
    to_physical_layer(&s);              /* indítsuk útnak */
    start_timer(s.seq);                 /* ha a válasz túl sokáig tart, legyen időtúllépés */
    wait_for_event(&event);             /* frame_arrival, cksum_err, timeout */
    if (event == frame_arrival) {
      from_physical_layer(&s);          /* szerezzük meg a nyugtát */
      if (s.ack == next_frame_to_send) {
        from_network_layer(&buffer);    /* szerezzük meg a következő elküldeni valót */
        inc(next_frame_to_send);        /* invertáljuk a next_frame_to_send-et */
      }
    }
  }
}

void receiver3(void)
{
  seq_nr frame_expected;
  frame r, s;
  event_type event;

  frame_expected = 0;
  while (true) {
    wait_for_event(&event);             /*lehetőségek: frame_arrival, cksum_err */
    if (event == frame_arrival) {       /* egy érvényes keret érkezett */
      from_physical_layer(&r);          /* szerezzük meg az újonnan érkezett keretet */
      if (r.seq == frame_expected) {    /* ez az amire vártunk */
        to_network_layer(&r.info);      /* adjuk tovább az adatot a hálózati rétegnek */
        inc(frame_expected);            /* legközelebb a másik sorszámot várjuk */
      }
      s.ack = 1 - frame_expected;       /* mondjuk meg, hogy melyik keretet nyugtázzuk */
      to_physical_layer(&s);            /* semelyik másik mezőt sem használjuk */
    }
  }
}
```

3.11. ábra. *A „pozitív nyugtázás újraküldéssel" protokoll*

megakadályozzuk, be kell vezetnünk valamilyen mechanizmust, amivel a küldő állomás tudomására hozzuk, hogy melyik keretet nyugtázta a vevő. A legegyszerűbb megközelítés az, hogy sorszámot teszünk a nyugtákba, amely azt mutatja, hogy melyik keretet nyugtázza az adott nyugta.

A 3. protokoll abban különbözik elődeitől, hogy az adónak és a vevőnek is van olyan változója, amelynek emlékszik az értékére, mialatt az adatkapcsolati réteg várakozási állapotban van. Az adó a következő elküldendő keret sorszámát tárolja a *next_frame_to_send*-ben; a vevő a következő venni kívánt keret sorszámát a *frame_expexted*-ben. Mindkét eljárásnak van egy rövid inicializációs fázisa, mielőtt belép a végtelen ciklusba.

Az adó állomás egy keret továbbítása után elindít egy időzítőt. Ha már ment az időzítő, akkor nullázódik, hogy megint egy teljes időzítési intervallum álljon rendelkezésre. Az időzítési intervallumot úgy kell megválasztani, hogy elég ideje legyen a keretnek eljutni a vevőhöz, a vevőnek feldolgozni azt a legrosszabb esetben is, és a nyugtakeretnek visszaérni az adó állomáshoz. Csak ha ez az idő eltelt, akkor feltételezhetjük biztonsággal, hogy vagy a továbbított keret vagy annak nyugtája elveszett, és csak ekkor küldhetünk egy másik példányt belőle.

Egy keret továbbítása és az időzítő elindítása után az adó állomás várakozni kezd, hátha történik valami izgalmas. Három lehetőség van: egy nyugtakeret érkezik sértetlenül, egy sérült nyugtakeret támolyog be vagy az időzítő lejár. Ha egy érvényes nyugta jön be, az adó lekéri a következő csomagot a hálózati rétegétől, és beleteszi a pufferbe, felülírva az előző csomagot, ezen kívül a sorszámot is lépteti. Ha sérült keret vagy egyáltalán semmi sem érkezik, sem a puffer, sem a sorszám nem változik, így a következő ciklusban egy újabb példányát tudjuk elküldeni az előbbi keretnek.

Amikor egy érvényes keret érkezik a vevőhöz, megvizsgáljuk a sorszámát, hogy meglássuk, duplikátum-e. Ha nem az, akkor elfogadjuk, továbbadjuk a hálózati rétegnek, és egy nyugtát küldünk. A duplikátumokat és a sérült kereteket nem adjuk át a hálózati rétegnek.

3.4. Csúszóablakos protokollok

Az eddigi protokollokban adat keretek csak az egyik irányba haladtak. A legtöbb gyakorlati helyzetben mindkét irányban kell adatokat átvinni. A duplex kapcsolat elérésének egyik módja az, hogy veszünk két különálló kommunikációs csatornát, és mind a kettőt szimplex adatforgalomra használjuk (különböző irányban). Ha ezt tesszük, lesz két különálló fizikai áramkörünk, melynek van egy „előre" csatornája (adatok számára) és egy „vissza" csatornája (nyugták számára). A visszirányú csatorna sávszélessége mindkét esetben majdnem teljesen kihasználatlanul marad. Valójában a felhasználó két áramkörért fizet, de csak egynek a kapacitását használja.

Jobb ötlet ugyanazt az áramkört használni mindkét irányra. Végül is a 2. és 3. protokollban már mindkét irányba továbbítottunk kereteket ugyanazzal az áramkörrel, és a visszirányú csatornának ugyanakkora kapacitása van, mint az előreirányúnak. Ebben a modellben az *A*-tól *B* felé tartó adatkeretek összekeverednek a *B*-től *A* felé menő

nyugtakeretekkel. A vevő a *kind* mező alapján tudja megmondani, hogy a keret adat- vagy nyugtakeret-e.

Bár az adat- és a nyugtakeretek egyetlen áramkörrel való átlapolt továbbítása már fejlődés ahhoz képest, mintha két különálló fizikai áramkörünk lenne, még van további fejlődési lehetőség. Amikor egy adatkeret megérkezik, ahelyett, hogy azonnal küldene egy külön vezérlő keretet, a vevő türtőzteti magát, és megvárja, hogy a hálózati réteg átadja neki a következő csomagot. A nyugtát hozzácsatolja a kimenő adatkerethez (a fejrész *ack* mezőjét használva). Valójában a nyugta így ingyen utazik a következő kimenő adatkerettel. Az a módszer, amikor a kimenő nyugtákat átmenetileg késleltetjük, hogy rá tudjuk akasztani a következő kimenő adatkeretre, **ráültetésként (piggybacking)** ismert.

A ráültetés alkalmazásának fő előnye a különálló nyugtakeretekhez képest a csatorna rendelkezésre álló sávszélességének jobb kihasználása. Az *ack* mező a keret fejrészében csak néhány bitbe kerül, míg egy külön kerethez kellene saját fejrész, a nyugta maga és ellenőrző összeg. Ráadásul a kevesebb elküldött keret kevesebb „keret érkezett" megszakítást jelent, és talán kisebb puffer igényt a vevőben attól függően, hogy hogyan van szervezve a szoftver. A következő protokollban, amit megvizsgálunk, a ráültetés mező csak egy bitet foglal el a keret fejrészében. Egyéb esetekben is ritkán foglal többet néhány bitnél.

A ráültetés azonban olyan komplikációt okoz, ami a különálló nyugtáknál nem jelentkezett. Meddig várakozzon az adatkapcsolati réteg a csomagra, amelyre ráülteti a nyugtát? Ha az adatkapcsolati réteg tovább vár, mint a küldő időzítési intervalluma, a küldő állomás a keretet újraküldi, meghiúsítva a nyugta célját. Ha az adatkapcsolati réteg jós lenne, és meg tudná jósolni a jövőt, meg tudná mondani, hogy mikor fog bejönni a következő csomag a hálózati rétegtől, és attól függően, hogy milyen hosszú a várható várakozás, el tudná dönteni, hogy várjon-e, vagy azonnal küldjön egy külön nyugtakeretet. Természetesen az adatkapcsolati réteg nem tudja megjósolni a jövőt, így valamilyen ad hoc elgondolásra kell hagyatkoznia, mint például, hogy várjon rögzített számú milliszekundumot. Ha az új csomag hamar megérkezik, a nyugtát ráültetjük; ha viszont nem érkezik új csomag az időzítési intervallum végéig, az adatkapcsolati réteg elküld egy önálló nyugtakeretet.

A 3. protokoll nemcsak szimplex volt, hanem különleges feltételek mellett, melyek korai időtúllépést eredményeztek, hibázott is. Jobb lenne, ha olyan protokollunk lenne, amelyik nem esik ki a szinkronból a tönkrement, elveszett keretek és az idő előtti időtúllépések semmilyen kombinációja ellenére sem. A következő három protokoll robusztusabb, és kóros feltételek mellett is tud működni. Mind a három a **csúszóablakos (sliding window)** protokollok osztályába tartozik. Amint később látni fogjuk, a három protokoll a hatékonyság, a bonyolultság és a puffer igény tekintetében különbözik egymástól.

Minden csúszóablakos protokollban minden kimenő keret tartalmaz egy 0 és valamilyen maximális érték közötti sorszámot. A maximum általában 2^n-1, így a sorszám szépen elfér egy n bites mezőben. A megáll-és-vár csúszóablakos protokoll $n = 1$-et használ, a sorszámokat 0-ra és 1-re korlátozva, de a kifinomultabb változatok tetszőleges n-t használhatnak.

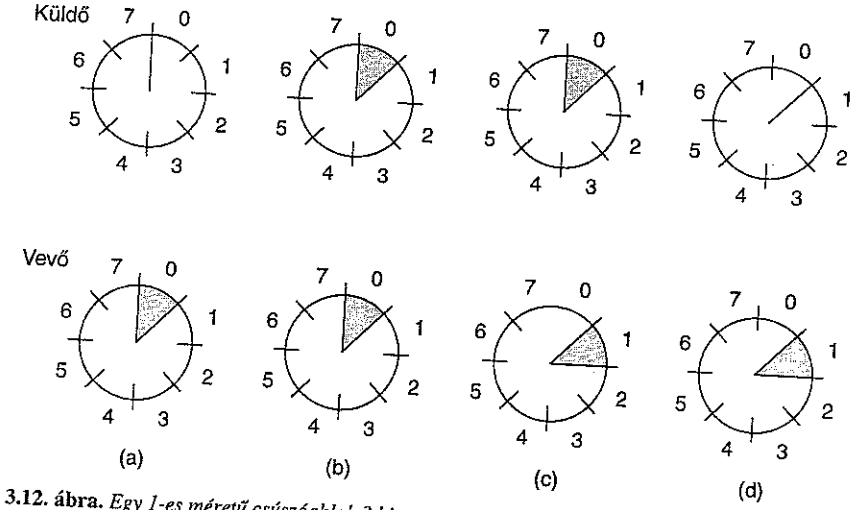
Minden csúszóablakos protokoll lényege az, hogy az adó állomás folyamatosan

karbantart egy sorszámhalmazt, amely az elküldhető kereteknek felel meg. Azt mondjuk, hogy ezek a keretek az **adási ablakba (sending window)** esnek. Hasonlóan a vevő is karbantart egy **vételi ablakot (receiving window)**, amely azoknak a keretek nek felel meg, amelyeket vehet. Az adó és a vevő ablakainak nem kell azonos alsó és felső határral rendelkeznie, sőt a méretének sem kell megegyeznie. Néhány protokollban az ablakok rögzített méretűek, másokban azonban nőhetnek vagy csökkenhetnek, ahogyan a kereteket küldjük és vesszük.

Bár ezek a protokollok több szabadságot adnak az adatkapcsolati rétegnek a kere tek küldésének és vételének sorrendjére vonatkozóan, továbbra sem vetjük el azt az igényt, hogy a protokollnak ugyanolyan sorrendben kell kézbesítenie a csomagokat a hálózati réteghez, mint amilyenben az adatkapcsolati réteg átvette azokat a küldő gépen. Szintén nem változtattuk meg azt az igényünket, hogy a fizikai kommunikációs csatorna „vezetékszerű" legyen, azaz ugyanolyan sorrendben kell a kereteket kézbesítenie, amilyenben elküldtük azokat.

A küldő ablakában levő sorszámok azokat a kereteket reprezentálják, amelyeket elküldtünk, de még nincsenek nyugtázva. Valahányszor egy új csomag érkezik a hálózati rétegtől, a következő legnagyobb sorszámot adjuk neki, és az ablak felső szélét eggyel továbbléptetjük. Ilyen módon, az ablak folyamatosan karbantartja a nyugtázatlan keretek listáját.

Mivel azok a keretek, amelyek a küldő ablakában végső soron elveszhetnek vagy megsérülhetnek az átvitel során, a küldő állomásnak az összes ilyen keretet a memóriájában kell tartani az esetleges újraküldések miatt. Ezért, ha a maximális ablakméret n, a küldőnek n pufferre van szüksége a nyugtázatlan keretek megtartásához. Ha az ablak eléri a maximális méretét, az adatkapcsolati rétegnek erőszakkal le kell kapcsolnia a hálózati réteget, amíg egy puffer fel nem szabadul.



3.12. ábra. *Egy 1-es méretű csúszóablak 3 bites sorszámmal. (a) Kezdetben. (b) Az első keret elküldése után. (c) Az első keret vétele után. (d) Az első nyugta vétele után*

A vevő adatkapcsolati rétegének ablaka azokhoz a keretekhez tartozik, amelyeket az adatkapcsolati réteg elfogadhat. Bármelyik keretet, ami kívül esik ezen az ablakon, minden további nélkül eldobunk. Amikor egy olyan keret érkezik, melynek a sorszáma egyenlő az ablak alsó szélével, átadjuk a hálózati rétegnek, nyugtát küldünk és az ablakot eggyel elforgatjuk. A vevő ablakának mérete – ellentétben a küldőével – mindig a kezdeti értéken marad. Megjegyezzük, hogy az 1 méretű ablak azt jelenti, hogy az adatkapcsolati réteg csak sorrendben fogadja el a kereteket, de nagyobb ablakoknál ez nem így van! A hálózati rétegbe azonban mindig a megfelelő sorrendben tápláljuk be a csomagokat, tekintet nélkül az adatkapcsolati réteg ablakának méretére.

A 3.12. ábrán egy példa látható az 1-es maximális ablakméretre. Kezdetben egyetlen keret sincs kinn, így az adó ablakának alsó és felső szélei egyenlőek, de ahogy az idő telik, a helyzet az ábrán látható módon változik.

3.4.1. Egybites csúszóablakos protokoll

Az általános esettel való megbirkózás előtt vizsgáljunk meg egy olyan csúszóablakos protokollt, amelynek maximum 1 nagyságú ablaka lehet. Egy ilyen protokoll a megáll-és-vár technikát alkalmazza, mivel a küldő állomás elküld egy keretet és megvárja ennek nyugtáját, mielőtt a következőt elküldené.

A 3.13. ábra egy ilyen protokollt ábrázol. Hasonlóan a többihez, ez is néhány változó definiálásával kezdődik. A *next_frame_to_send* azt mutatja meg, hogy éppen melyik keretet próbálja a küldő állomás elküldeni. Hasonlóan, a *frame_expected* azt mutatja meg, hogy a vevő melyik keretet várja. Mindkét változónál csak 0 és az 1 érték jöhet szóba.

Rendszerint a két adatkapcsolati réteg közül az egyik korábban kezd adni. Ezért csak az egyik adatkapcsolati réteg programjának kellene tartalmaznia a *to_physical_layer* és a *start_timer* eljáráshívásokat a fő cikluson kívül. Abban az esetben, ha mindkét adatkapcsolati réteg egyszerre kezd adni, sajátos helyzet alakul ki, melyet később tárgyalunk meg. A kezdő gép lekéri az első csomagot a hálózati rétegétől, egy keretet épít belőle, és elküldi azt. Amikor ez a keret (vagy bármelyik) megérkezik, a vevő adatkapcsolati réteg – pontosan úgy, mint a 3. protokollban – megnézi, hogy duplikátum-e. Ha a keret az, amelyiket várt, átadja a hálózati rétegnek, és a vevő ablakát feljebb csúsztatja.

A nyugta mező a legutolsó hibátlanul vett keret sorszámát tartalmazza. Ha ez a szám megegyezik annak a keretnek a sorszámával, amit az adó próbál elküldeni, az adó tudja, hogy végzett a *buffer*-ben tárolt csomaggal, és lekérheti a következőt a hálózati rétegtől. Ha a sorszám nem egyezik, folytatnia kell a próbálkozást ugyanazzal a kerettel. Minden alkalommal, amikor egy keret érkezik, egyet el is küldünk.

Most vizsgáljuk meg a 4. protokollt, hogy lássuk, mennyire ellenálló, kóros eseménysorozatok esetén! Tételezzük fel, hogy A a 0-s keretét próbálja elküldeni B-nek, és B szintén a 0-s keretét próbálja elküldeni A-nak. Tegyük fel, hogy A elküldi a keretet B-nek, de az időzítési intervalluma egy kicsit rövidebb, mint kellene. Ezért több ízben lejárhat az időzítője, így egy sor azonos keretet küldhet, mindet $seq = 0$-val és $ack = 1$-gyel.

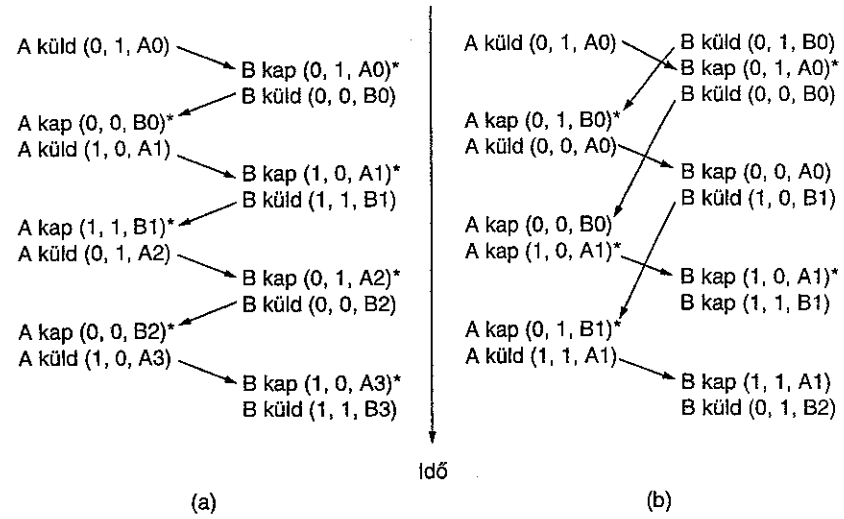
```
/* A 4. protokoll (csúszóablakos) kétirányú és robusztusabb, mint a 3. protokoll. */

#define MAX_SEQ 1                       /* 1-nek kell lennie a 4. protokollhoz */
typedef enum {frame_arrival, cksum_err, timeout} event_type;
#include "protocol.h"

void protocol4(void)
{
  seq_nr next_frame_to_send;            /* csak 0 vagy 1 lehet */
  seq_nr frame_expected;                /* csak 0 vagy 1 lehet */
  frame r, s;                           /* pufferek keretnek */
  packet buffer;                        /* az éppen küldendő csomag */
  event_type event;

  next_frame_to_send = 0;               /* következő keret a kimenő folyamban */
  frame_expected = 0;                   /* a várt érkező keret száma */
  from_network_layer(&buffer);          /* kérjünk le egy csomagot a hálózati rétegtől*/
  s.info = buffer;                      /* készítsük elő a kezdő keretet az elküldéshez */
  s.seq = next_frame_to_send;           /* tegyük bele a sorszámot a keretbe */
  s.ack = 1 - frame_expected;           /* ráültetett nyugta */
  to_physical_layer(&s);                /* továbbítsuk a keretet */
  start_timer(s.seq)                    /* indítsuk el az időzítőt */

  while (true) {
    wait_for_event(&event);             /* frame_arrival, cksum_err vagy timeout */
    if (event == frame_arrival) {       /* egy keret érkezett sértetlenül */
      from_physical_layer(&r);          /* szerezzük meg */

      if (r.seq == frame_expected) {
        /* kezeljük a bejövő keretfolyamot */
        to_network_layer(&r.info);      /* adjuk tovább a csomagot a hálózati rétegnek */
        inc(frame_expected);            /* invertáljuk a legközelebb várt sorszámot */
      }
      if (r.ack == next_frame_to_send) { /* kezeljük a kimenő keretfolyamot */
        from_network_layer(&buffer);    /* kérjük le a következő csomagot a hálózati
                                           rétegtől */
        inc(next_frame_to_send);        /* invertáljuk a küldő sorszámát */
      }
    }
    s.info = buffer;                    /* készítsük el a kimenő keretet */
    s.seq = next_frame_to_send;         /* tegyük bele a keretbe a sorszámot */
    s.ack = 1 - frame_expected;         /* a legutóbb vett keret sorszáma */
    to_physical_layer(&s);              /* továbbítsuk a keretet */
    start_timer(s.seq);                 /* indítsuk el az időzítőt */
  }
}
```

3.13. ábra. *Egybites csúszóablakos protokoll*

Amikor az első érvényes keret megérkezik *B*-be, az elfogadja és a *frame_expected*-et 1-re állítja. Az összes ezt követő keretet elutasítja, mert *B* most olyan keretet vár, melynek a sorszáma 1, nem pedig 0. Továbbá, mivel minden duplikátumnál *ack* = 1 és *B* még 0-s nyugtára vár, *B* nem fog lekérni újabb csomagot a hálózati rétegétől.

Miután mindegyik elutasítandó duplikátum megérkezett, *B* küld *A*-nak egy keretet *seq* = 1-gyel és *ack* = 0-val. Végül is ezek közül az egyik megérkezik *A*-hoz, és *A* elkezdi küldeni a következő csomagot. Az elveszett keretek vagy idő előtti időtúllépések semmilyen kombinációja nem tudja azt okozni, hogy a protokoll akár megkettőzött csomagokat küldjön a hálózati rétegnek, akár kihagyjon csomagot, vagy holtpontba kerüljön.

Egy sajátos helyzet alakul azonban ki, ha mindkét oldal egyszerre küldi el a kezdeti csomagot. Ez a szinkronizációs probléma látható a 3.14. ábrán. Az (a) részben a protokoll normális működése figyelhető meg. A (b) rész a sajátos helyzetet illusztrálja. Ha *B* megvárja *A* első keretét, mielőtt a sajátjai közül egyet elküldene, az eseménysorozat olyan, mint az (a)-n látható, és minden keretet elfogadunk. Ha azonban *A* és *B* egyszerre indítja el a kommunikációt, az első kereteik keresztezik egymást, és az adatkapcsolati réteg a (b)-n látható helyzetbe kerülhet. Az (a) ábrán minden keret érkezése új csomagot hoz a hálózati rétegnek, nincsenek duplikátumok. A (b)-n a keretek fele duplikátumot tartalmaz, még akkor is, ha nincsenek átviteli hibák. Hasonló helyzetek alakulhatnak ki idő előtti időtúllépések eredményeként, még akkor is, ha az indulásnál egyértelműen csak az egyik állomás kezd adni. Valójában, ha többszörös korai időtúllépés következik be, a kereteket háromszor vagy még többször is elküldheti a protokoll.

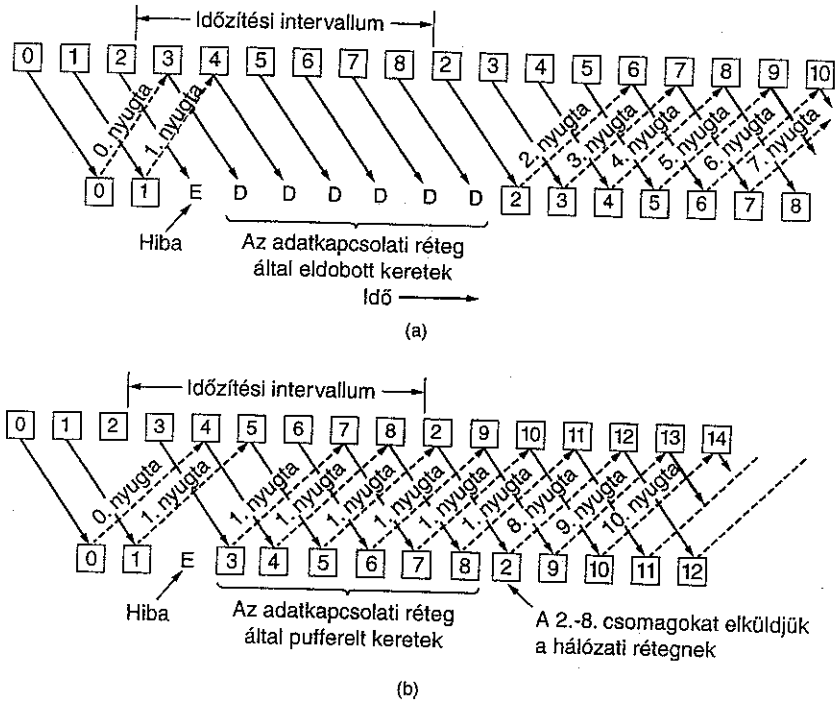


3.14. ábra. *Két eseménysorozat a 4. protokollhoz. A jelölés: sorszám, nyugtaszám, csomagsorszám. A csillag azt jelzi, hogy az adatkapcsolati réteg elfogadta a keretet*

3.4.2. Az n visszalépést alkalmazó protokoll

Az eddigiekben azzal a hallgatólagos feltételezéssel éltünk, hogy egy keret vevőhöz való megérkezéséhez és a nyugta visszaérkezéséhez együttesen szükséges idő elhanyagolható. Néha ez a feltételezés egyáltalán nem helytálló. Ezekben a szituációkban a hosszú körülfordulási idő fontos következményeket von maga után a sávszélesség kihasználtságára nézve. Példának okáért, vegyünk egy 50 kb/s-os műholdas csatornát 500 ms-os körülfordulási idővel. Képzeljük el, hogy a 4. protokollt szeretnénk használni 1000 bites keretek küldésére a műholdon keresztül. A $t = 0$ időpillanatban az adó állomás elkezdi küldeni az első keretet. A $t = 20$ ms-ban már a teljes keretet elküldte. Nem előbb, mint $t = 270$ ms érkezik meg a teljes keret a vevőhöz, és nem előbb, mint $t = 520$ ms érkezik meg a nyugta az adó állomáshoz a legjobb esetben (nincs várakozás a vevőben, és rövid a nyugtakeret). Ez azt jelenti, hogy az adó az idő 500/520 részében, azaz 96%-ában blokkolva van (tehát a rendelkezésre álló sávszélességnek csupán a 4%-át használja ki). Tisztán látszik, hogy a hosszú átviteli idő, a nagy sávszélesség és a rövid keretek kombinációja végzetes a hatékonyságra nézve.

A fent leírt problémát azon szabály következményének tekinthetjük, amely megkö-



3.15. ábra. *(a) Egy hiba hatása, ha a vételi ablak mérete 1. (b) Egy hiba hatása, ha a vételi ablak nagy*

veteli az adó állomástól, hogy megvárja az elküldött keret nyugtáját, mielőtt egy másik keretet küldene. Ha lazítunk ezen a korlátozáson, sokkal nagyobb hatékonyságot érhetünk el. A megoldás alapvetően arra épül, hogy megengedjük a küldőnek, hogy 1 helyett legfeljebb w darab keretet elküldjön, mielőtt blokkolódik, így w megfelelő megválasztásával az adó állomás folyamatosan küldeni tudja a kereteket a körülfordulási idővel azonos ideig anélkül, hogy betelne az ablak. A fenti példában w legalább 26 kell hogy legyen. Az adó állomás elkezdi küldeni a 0. keretet, mint az előbb. Akkor, amikor befejezte a 26 keret elküldését $t = 520$ ms-nál, a 0. keret nyugtája éppen megérkezett. Ezek után minden 20 ms-ban érkezik egy nyugta, így az adó mindig megkapja az engedélyt a folytatásra, épp akkor, amikor kell. Így minden időpillanatban 25 vagy 26 nyugtázatlan keret van kinn, azaz az adó ablakának maximális mérete 26.

Ez a technika **csővezetékezésként (pipelining)** ismert. Ha a csatornakapacitás b bit/s, a kerethossz l bit és a körülfordulási idő R s, egyetlen keret továbbításához szükséges idő l/b s. Miután az adatkeret utolsó bitjét is elküldtük, az $R/2$ s múlva ér oda a vevőhöz, és még legalább $R/2$ s kell a nyugtának, hogy visszaérjen az adó állomáshoz. Ez összesen R s késleltetés. A megáll-és-vár technikánál a vonal l/b-ig foglalt, és R-ig üresjáratban van, ami $l/(l + bR)$ vonalkihasználtságot jelent. Ha $l < bR$, a hatékonyság kisebb lesz mint 50%. Mivel midig van valamilyen nullától különböző késleltetési idő a nyugta visszaérkezéséig, elvileg a csővezetékezés használható arra, hogy a vonalat ez idő alatt is kihasználjuk. Ha azonban a késleltetés kicsi, a nagyobb bonyolultság miatt ezen technika alkalmazása nem éri meg a fáradságot.

A csővezetékezés alkalmazása megbízhatatlan csatornán néhány komoly kérdést vet fel. Először is: mi történik, ha egy hosszú folyam közepén egy keret megsérül, vagy elveszik? Mielőtt az adó egyáltalán észrevenné, hogy valami nincs rendben, nagyszámú további keret érkezik meg a vevőhöz. Amikor egy sérült keret érkezik a vevőhöz, nyilvánvalóan el kell dobnia, de mit csináljon az azt követő hibátlan keretekkel? Ne feledkezzünk meg róla, hogy a vevő adatkapcsolati rétegének mindenképpen a helyes sorrendben kell a hálózati rétegnek átadni a csomagokat.

A csővezetékezésnél a fellépő hibák kezelésének két alapvető megközelítése van. Az egyik megközelítést **„visszalépés n-nel" (go back n)** eljárásnak nevezik. Ekkor a vevő eldobja az összes keretet, amely a hibás után érkezik, és nem küld nyugtát róluk. Ez a stratégia az 1 hosszúságú vételi ablaknak felel meg. Más megközelítésből, az adatkapcsolati réteg elutasít minden keretet, kivéve a soron következőt, amit a hálózati rétegnek át kell adnia. Ha az adó ablaka betelik mielőtt az időzítő lejár, a csővezeték elkezd kiürülni. Végül is az adónak lejár az időzítője és újraküldi az összes nyugtázatlan keretet, kezdve a sérült, vagy elveszett kerettel. Ez a megközelítés, mely a 3.15.(a) ábrán látható, nagy sávszélességet pazarolhat el, ha nagy a hibaarány.

A másik általános stratégia a hibák kezelésére, amikor csővezetékezést alkalmazunk, a **szelektív ismétlés (selective repeat).** Ekkor a vevő adatkapcsolati rétege tárolja az összes hibátlan keretet, amelyik a hibás után jön. Amikor végül az adó észreveszi, hogy valami nem stimmel, csak újraküldi az egyetlen rossz keretet – de nem az összes azt követőt. Ez a módszer a 3.15.(b) ábrán látható. Ha a második próbálkozás sikeres, a vevő adatkapcsolati rétegnek egyszerre sok hibátlan kerete lesz helyes sorrendben, így azokat gyorsan át tudja adni a hálózati rétegnek, és a legnagyobb sorszámú keretet nyugtázhatja.

```
/* Az 5. protokoll (csővezetékezéses) megengedi, hogy több keret legyen úton. A küldő
   leadhat MAX_SEQ darab keretet anélkül, hogy nyugtára várna. Ráadásul nem
   feltételezzük a hálózati rétegről, hogy mindig van új csomagja. Ehelyett a hálózati
   réteg network_layer_ready eseményt okoz, amikor van elküldeni való kerete.*/

#define MAX_SEQ 7                       /* 2^n – 1-nek kell lennie */
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready} event_type;
#include "protocol.h"

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* true-t ad vissza, ha a <= b < c körkörösen; egyébként false-t */
  if (((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a)))
      return(true);
    else
      return(false);
}

static void send_data(seq_nr frame_nr, seq_nr frame_expected, packet buffer[])
{
/* Építsünk fel és küldjünk el egy adatkeretet. */
  frame s;                              /* átmeneti változó */

  s.info = buffer[frame_nr];            /* tegyük be a csomagot a keretbe */
  s.seq = frame_nr;                     /* tegyük be a sorszámot a keretbe */
  s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);   /* ültessük rá a nyugtát */
  to_physical_layer(&s);                /* továbbítsuk a keretet */
  start_timer(frame_nr);                /* indítsuk el az időzítőt */
}

void protocol5(void)
{
  seq_nr next_frame_to_send;            /* MAX_SEQ > 1; a kimenő keretekhez használjuk *
  seq_nr ack_expected;                  /* a legrégebbi keret ami még nincs nyugtázva */
  seq_nr frame_expected;                /* a következő várt keret a bejövő folyamban */
  frame r;                              /* átmeneti változó */
  packet buffer[MAX_SEQ + 1];           /* pufferek a kimenő folyamhoz */
  seq_nr nbuffered;                     /* a használatban levő kimeneti pufferek */
  seq_nr i;                             /* a puffertömb indexelésére használjuk */
  event_type event;

  enable_network_layer(); /* engedélyezzük a network_layer_ready eseményeket */
  ack_expected = 0;                     /* a következőként várt bejövő nyugta */
  next_frame_to_send = 0;               /* következő kimenő keret */
  frame_expected = 0;                   /* a várt érkező keret száma */
  nbuffered = 0;                        /* kezdetben egyetlen csomag sincs pufferelve */

while (true) {
    wait_for_event(&event); /* négy lehetőség: lásd az event_type-ot fent */
```

3.16. ábra. *A „visszalépés n-nel" eljárást használó csúszóablakos protokoll*


```
    switch(event) {
      case network_layer_ready:     /* a hálózati rétegnek van elküldeni való
                                       csomagja */
        /* Fogadjunk, mentsünk el és továbbítsunk egy új keretet. */
        from_network_layer(&buffer[next_frame_to_send]); /* kérjünk le új csomagot */
        nbuffered = nbuffered + 1;      /* növeljük a küldő ablakát */
        send_data(next_frame_to_send, frame_expected, buffer); /* továbbítsuk a keretet */
        inc(next_frame_to_send);        /* léptessük a küldő ablakának felső szélét */
        break;

      case frame_arrival:               /* egy adat-, vagy vezérlőkeret érkezett */
        from_physical_layer(&r);        /* szerezzük meg a bejövő keretet a fizikai rétegtől */

        if (r.seq == frame_expected) {
          /* A kereteket csak sorrendben fogadjuk el. */
          to_network_layer(&r.info);    /* adjuk át a csomagot a hálózati rétegnek */
          inc(frame_expected);          /* léptessük a vevő ablakának alsó szélét */
        }

        /* Az n. nyugta magában foglalja az n - 1.-t, n - 2.-t, stb. Ellenőrizzük ezt. */
        while (between(ack_expected, r.ack, next_frame_to_send))
        {
          /* Kezeljük a ráültetett nyugtát. */
          nbuffered = nbuffered - 1;    /* eggyel kevesebb van pufferelve */
          stop_timer(ack_expected);     /* keret érkezett sértetlenül; állítsuk le az időzítőt */
          inc(ack_expected);            /* húzzuk összébb a küldő ablakát */
        }
        break;

      case cksum_err: break;            /* egyszerűen hagyjuk figyelmen kívül a hibás kereteket */

      case timeout:                     /* baj van; küldjük újra az összes kint levő keretet */
        next_frame_to_send = ack_expected;          /* kezdjük itt az újraküldést */
        for (i = 1; i <= nbuffered; i++) {
          send_data(next_frame_to_send, frame_expected, buffer);  /* küldjünk újra
                                                                     1 keretet */
          inc(next_frame_to_send);      /* készüljünk fel a következő küldésére */
        }
    }

    if (nbuffered < MAX_SEQ)
      enable_network_layer();
    else
      disable_network_layer();
  }
}
```

3.16. ábra. *A „visszalépés n-nel" eljárást használó csúszóablakos protokoll (folytatás)*

Ez a stratégia az 1-nél nagyobb vételi ablaknak felel meg. Bármelyik keretet, amelyik az ablakban van, még elfogadhatjuk, és emiatt pufferelni kell, amíg minden előzőt át nem adtunk a hálózati rétegnek. Ez a megoldás így sok memóriát igényelhet az adatkapcsolati rétegben.

Ez a két megközelítés lehetőséget ad arra, hogy kompromisszumot kössünk a sávszélesség-kihasználás és az adatkapcsolati rétegbeli memória mérete között. Attól függően, hogy melyik erőforrás az értékesebb, az egyik vagy a másik megközelítés használható. A 3.16. ábrán egy olyan csővezetékezést alkalmazó protokoll látható, amelyben az adatkapcsolati réteg csak sorrendben fogadja el a kereteket; egy hibás keretet követően az összes keretet eldobja. Ebben a protokollban – első ízben – szakítottunk azzal a feltételezéssel, hogy a hálózati rétegnek végtelen forrása van csomagokból. Amikor a hálózati rétegnek van egy kerete, amit el akar küldeni, *network_layer_ready* eseményt okozhat. Annak érdekében azonban, hogy megfeleljünk a forgalomszabályozási megkötésnek, mely szerint semmikor sem lehet kint több mint *MAX_SEQ* darab nyugtázatlan keret, az adatkapcsolati rétegnek le kell tudni tiltani a hálózati réteget, hogy az több munkával zaklassa. Az *enable_network_layer* és a *disable_network_layer* könyvtári rutinok végzik ezt a feladatot.

Vegyük észre, hogy minden időpillanatban legfeljebb *MAX_SEQ* és nem *MAX_SEQ* + 1 darab keret lehet kint, annak ellenére, hogy *MAX_SEQ* + 1 különböző sorszám van: 0, 1, 2, ..., *MAX_SEQ*. Hogy megértsük, miért szükséges ez a korlátozás, vegyük a következő eseménysort *MAX_SEQ* = 7 mellett.

1. Az adó állomás elküldi a kereteket a 0.-tól a 7.-ig.

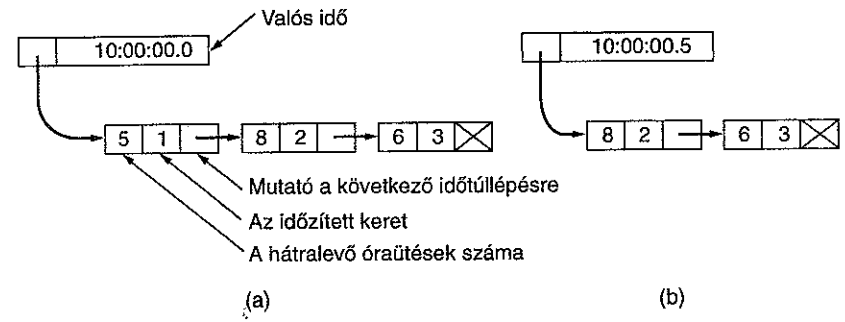
2. A 7. keretre végül is visszajön a küldő állomáshoz egy ráültetett nyugta.

3. Az adó állomás újabb nyolc keretet küld, megint 0 és 7 közötti sorszámokkal.

4. Most még egy ráültetett nyugta érkezik a 7. keretre.

A kérdés: a második sorozathoz tartozó nyolc keret mind megérkezett sikeresen, vagy mind a nyolc elveszett (a hibás keretet követő keretek eldobását is keretvesztésnek számítva). A vevőnek mindkét esetben a 7. keretre kellene nyugtát küldenie. Az adó állomás semmilyen módon nem tudja a kérdést eldönteni. Emiatt a kint levő keretek száma legfeljebb *MAX_SEQ* lehet.

Bár az 5. protokoll nem puffereli az érkező kereteket egy hibás keret után, mégsem ússza meg teljesen a pufferelés problémáját. Mivel az adó állomásnak lehet, hogy újra kell küldeni az összes nyugtázatlan keretet egy későbbi időpontban, meg kell tartania az összes elküldött keretet addig, amíg biztosan nem tudja, hogy a vevő elfogadta őket. Amikor nyugta jön az *n*. keretre, az $n-1$., $n-2$. stb. szintén automatikusan nyugtázódik. Ez a tulajdonság különösen fontos akkor, amikor néhány előző nyugtahordozó keret elveszik vagy tönkremegy. Mindig amikor egy nyugta beérkezik, az adatkapcsolati réteg ellenőrzi, hogy fel lehet-e szabadítani valamelyik puffert. Ha fel lehet szabadítani puffereket (azaz van hely az ablakban), a korábban blokkolt hálózati rétegnek engedélyezni lehet, hogy további *network_layer_ready* eseményeket okozzon.



3.17. ábra. *Több időzítő szimulációja szoftverből*

Mivel ebben a protokollban több kint levő keret van, logikailag több időzítőre van szükség: minden kint levő kerethez egyre. Minden keret időzítője az összes többiétől függetlenül jár le. Az összes időzítő könnyedén szimulálható szoftverből egy olyan hardver időzítővel, amelyik periodikusan megszakításokat okoz. A járó időzítők egy láncolt listát alkotnak, melyben minden csomópont azt mutatja, hogy mennyi óraütés van hátra, amíg az időzítő lejár, és hogy melyik kerethez tartozik az adott időzítő. A csomópont ezen kívül tartalmaz még egy mutatót is a következő csomópontra.

Egy illusztrációként arra, hogy hogyan lehet megvalósítani az időzítőket, vegyük a 3.17. ábra példáját. Tételezzük fel, hogy az óra minden 100 ms-ban üt egyet. Kezdetben a valós idő 10:00:00.0, és három függőben levő időzítés van: 10:00:00.5-kor, 10:00:01.3-kor és 10:00:01:9-kor. Mindig, amikor a hardver óra üt, a valós időt aktualizáljuk, és az ütésszámlálót a lista fejében csökkentjük eggyel. Amikor az ütésszámláló nulla lesz, az időzítő időtúllépést okoz, és a csomópontot eltávolítjuk a listából, ahogyan a 3.17.(b) ábrán látható. Bár ez a szervezés azt kívánja, hogy a listát végignézzük, amikor a *start_timer*-t vagy a *stop_timer*-t meghívjuk, óraütésenként nem kell sok mindent csinálni. Az 5. protokollban mindkét rutinnak egy paramétert adtunk át, mely azt jelzi, hogy melyik kerethez tartozik az időzítő.

3.4.3. Szelektív ismétlést alkalmazó protokoll

Az 5. protokoll jól működik akkor, ha a hibák ritkák, de ha a vonal gyenge minőségű, nagy sávszélességet elpazarol a keretek újraküldésére. Egy másik stratégia a hibák kezelésére az, hogy megengedjük a vevőnek, hogy elfogadja és pufferelje a kereteket, melyek egy elveszett vagy megsérült keretet követnek. Az ilyen protokoll nem dob el kereteket csak azért, mert egy korábbi keret megsérült vagy elveszett.

Ebben a protokollban mind az adó, mind a vevő karbantart egy ablakot az elfogadható sorszámokból. Az adó ablakának mérete 0-ról indul és valamilyen előre definiált maximumig (*MAX_SEQ*) növekszik. A vevő ablaka ezzel ellentétben rögzített méretű, *MAX_SEQ*-val egyenlő. A vevőnek minden az ablakában levő sorszámhoz fenntartott puffere van. Mindegyik pufferhez tartozik egy bit (*arrived*), mely azt mutatja, hogy a

puffer tele van-e vagy üres. Amikor egy keret megérkezik, a *between* függvénnyel ellenőrizzük a sorszámát, hogy beleesik-e az ablakba. Ha beleesik és a keretet még nem vettük korábban, akkor elfogadjuk és tároljuk. Ezt tekintet nélkül arra, hogy a keret azt a csomagot tartalmazza-e, amit a hálózati réteg vár, vagy sem, mindig megteszszük. Természetesen, a keretet meg kell őrizni az adatkapcsolati rétegben, és nem szabad átadni a hálózati rétegnek addig, amíg az összes alacsonyabb sorszámú keretet nem továbbítottuk a hálózati rétegnek a helyes sorrendben. Az ezen algoritmust használó protokoll a 3.18. ábrán látható.

```
/* A 6. protokoll (nem sorrendhelyes vétel) elfogadja a nem soron következő kereteket
   is, de a csomagokat sorrendben továbbítja a hálózati rétegnek. Minden úton levő
   kerethez tartozik egy időzítő. Amikor az időzítő lejár, csak a hozzá tartozó keretet
   küldjük újra, nem az összeset, mint az 5. protokollban. */

#define MAX_SEQ 7                       /* 2^n - 1-nek kell lennie */
#define NR_BUFS ((MAX_SEQ + 1)/2)
typedef enum {frame_arrival, cksum_err, timeout, network_layer_ready, ack_timeout}
   event_type;
#include "protocol.h"
boolean no_nak = true;                  /* még nem küldtünk negatív nyugtát */
seq_nr oldest_frame = MAX_SEQ + 1; /* a kezdeti érték csak a szimulátor számára
                                      érdekes */

static boolean between(seq_nr a, seq_nr b, seq_nr c)
{
/* Ugyanaz, mint a between az 5. protokollban, de rövidebb és nehezebben érthető */
   return ((a <= b) && (b < c)) || ((c < a) && (a <= b)) || ((b < c) && (c < a));
}

static void send_frame(frame_kind fk, seq_nr frame_nr, seq_nr frame_expected,
   packet buffer[])
{
/* Építsünk fel és küldjünk el egy adat-, nyugta- vagy negatív nyugtakeretet. */
   frame s;                             /* átmeneti változó */

   s.kind = fk;                         /* kind == data, ack vagy nak */
   if (fk == data) s.info = buffer[frame_nr % NR_BUFS];
   s.seq = frame_nr;                    /* csak adatkereteknél van jelentése */
   s.ack = (frame_expected + MAX_SEQ) % (MAX_SEQ + 1);
   if (fk == nak) no_nak = false;       /* keretenként egy negatív nyugta */
   to_physical_layer(&s);               /* továbbítsuk a keretet */
   if (fk == data) start_timer(frame_nr % NR_BUFS);
   stop_ack_timer();                    /* nincs szükség külön nyugta keretre */
}

void protocol6(void)
{
```

3.18. ábra. *Szelektív ismétlést alkalmazó csúszóablakos protokoll*


```
   seq_nr ack_expected;                 /* a küldő ablakának alsó széle */
   seq_nr next_frame_to_send;           /* a küldő ablakának felső széle + 1 */
   seq_nr frame_expected;               /* a vevő ablakának alsó széle */
   seq_nr too_far;                      /* a vevő ablakának felső széle +1 */
   int i;                               /* index a pufferparkhoz */
   frame r;                             /* átmeneti változó */
   packet out_buf[NR_BUFS];             /* pufferek a kimenő folyamhoz */
   packet in_buf[NR_BUFS];              /* pufferek a bejövő folyamhoz */
   boolean arrived[NR_BUFS];            /* bejövő keretek bittérképe */
   seq_nr nbuffered;                    /* hány kimeneti puffer van jelenleg használatban */
   event_type event;
   enable_network_layer();              /* inicializálás */
   ack_expected = 0;                    /* a következőként várt bejövő nyugta */
   next_frame_to_send = 0;              /* következő kimenő keret */
   frame_expected = 0;
   too_far = NR_BUFS;
   nbuffered = 0;                       /* kezdetben egyetlen csomag sincs pufferelve */

   while (true) {
      wait_for_event(&event); /* öt lehetőség: lásd az event_type-ot fent */
      switch(event) {
         case network_layer_ready:      /* Fogadjunk, mentsünk el és továbbítsunk egy
                                           új keretet. */
            nbuffered = nbuffered + 1;  /* növeljük az ablakot */
            from_network_layer(&out_buf[next_frame_to_send %
               NR_BUFS]);               /* kérjünk le új csomagot */
            send_frame(data, next_frame_to_send, frame_expected,
               out_buf);                /* továbbítsuk a keretet */
            inc(next_frame_to_send);    /* léptessük a küldő ablakának felső szélét */
            break;

         case frame_arrival:            /* egy adat- vagy vezérlőkeret érkezett */
            from_physical_layer(&r);    /* kérjük le a bejövő keretet a fizikai rétegtől */
            if (r.kind == data) {
               /* Egy sértetlen keret érkezett. */
               if (r.seq != frame_expected && no_nak )
                  send_frame(nak, 0, frame_expected, out_buf); else start_ack_timer();
               if (between(frame_expected, r.seq, too_far) && (arrived[r.seq % NR_BUFS] == false)) {
                  /* A kereteket bármilyen sorrendben el lehet fogadni. */
                  arrived[r.seq % NR_BUFS] = true; /* jelöljük meg a puffert teliként */
                  in_buf[r.seq % NR_BUFS] = r.info; /* tegyük az adatot a pufferbe */
                  while (arrived[frame_expected % NR_BUFS]) {
                     /* Adjuk át a kereteket és léptessük az ablakot. */
                     to_network_layer(&in_buf[frame_expected % NR_BUFS]);
                     no_nak = true;
                     arrived[frame_expected % NR_BUFS] = false;
                     inc(frame_expected);   /* léptessük a vevő ablakának alsó szélét */
                     inc(too_far);          /* léptessük a vevő ablakának felső szélét */
```

3.18. ábra. *Szelektív ismétlést alkalmazó csúszóablakos protokoll (folytatás)*

```
                start_ack_timer();          /* hogy lássuk, hogy szükséges-e külön nyugta */
              }
            }
          }
          if ((r.kind == nak) && between(ack_expected, (r.ack + 1) % (MAX_SEQ + 1),
next_frame_to_send))
            send_frame(data, (r.ack + 1) % (MAX_SEQ + 1), frame_expected, out_buf);

          while (between(ack_expected, r.ack, next_frame_to_send)) {
            nbuffered = nbuffered - 1; /* kezeljük a ráültetett nyugtát. */
            stop_timer(ack_expected % NR_BUFS); /* ép keret érkezett */
            inc(ack_expected);          /* léptessük a küldő ablakának alsó szélét */
          }
          break;

        case cksum_err:
          if (no_nak) send_frame(nak, 0, frame_expected, out_buf); /* sérült keret */
          break;

        case timeout:
          send_frame(data, oldest_frame, frame_expected, out_buf); /* lejárt az
                                                                      időzítőnk */
          break;

        case ack_timeout:
          send_frame(ack, 0, frame_expected, out_buf); /* a nyugta időzítő lejárt,
küldjünk nyugtát */
      }

      if (nbuffered < NR_BUFFS) enable_network_layer(); else disable_network_layer();
    }
}
```

3.18. ábra. *Szelektív ismétlést alkalmazó csúszóablakos protokoll (folytatás)*

A keretek nem sorrendben történő vétele olyan problémákat vet fel, melyek nem jelentkeznek azokban a protokollokban, amelyekben a kereteket csak sorrendben fogadjuk el. Tételezzük fel, hogy 3 bites sorszámunk van, így az adó állomás legfeljebb hét keretet továbbíthat, mielőtt várakoznia kellene nyugtára. Kezdetben az adó és a vevő ablakai a 3.19.(a) ábrán láthatónak megfelelőek. Az adó állomás most továbbítja a kereteket a 0.-tól a 6.-ig. A vevő ablaka megengedi bármely keret elfogadását, melynek a sorszáma 0 és 6 közé esik, a határokat is beleértve. Mind a hét keret rendben megérkezik, így a vevő nyugtázza őket, és lépteti az ablakát, hogy lehetővé tegye a 7., 0., 1., 2., 3., 4., 5. és a 6. vételét. A beállt állapot a 3.19.(b) ábrán látható. Mind a hét puffert üresnek jelöljük meg.

Most érkeztünk el ahhoz a ponthoz, amikor beüt a mennykő egy villám formájában, mely a telefonoszlopot veszi célba, és kipucolja a vonalról az összes nyugtát. A küldő állomásnak előbb-utóbb lejár az időzítője, és újraadja a 0. keretet. Amikor ez a keret megérkezik a vevőhöz, a vevő ellenőrzi, hogy beleesik-e a vételi ablakba. Szerencsétlenségünkre a 3.19.(b) ábrán a 0. keret az új ablakon belül van, így elfogadjuk. A vevő egy ráültetett nyugtát küld a 6. keretről, hiszen a 0. és a 6. közötti kereteket vettük.

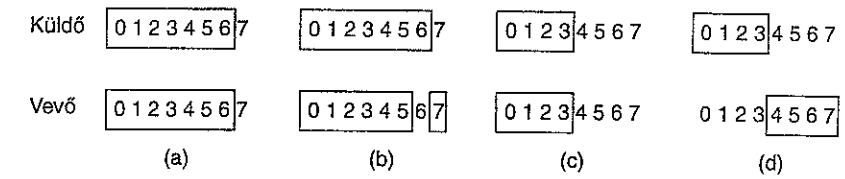
Az adó állomás örül, amikor megtudja, hogy mindegyik továbbított kerete rendben megérkezett, így lépteti az ablakát, és azonnal elküldi a 7., 0., 1., 2., 3., 4. és 5. keretet. A 7. keretet a vevő elfogadja, és a benne levő csomagot rögtön átadja a hálózati rétegnek. Mindjárt azután az adatkapcsolati réteg megnézi, hogy van-e már érvényes 0. kerete, rájön, hogy van, és továbbadja a benne levő csomagot a hálózati rétegnek. Következésképp a hálózati réteg helytelenül kap egy csomagot, a protokoll hibázik.

A probléma lényege az, hogy miután a vevő léptette az ablakát, az új érvényes sorszámok tartománya átlapolódik a régiekével. Az ezt követő csomag köteg tartalmazhat akár duplikátumokat (ha az összes nyugta elveszett), akár új kereteket (ha az összes nyugta megérkezett). A szegény vevő sehogyan sem tud különbséget tenni a két eset között.

A dilemmából kivezető megoldás azon alapul, hogy biztosnak kell lennünk abban, hogy miután a vevő léptette az ablakát, nincs átfedés az eredeti ablakkal. Ahhoz, hogy biztosítsuk az átfedésmentességet, a maximális ablakméret legfeljebb a sorszámok tartományának fele szabad kell legyen, mint ahogyan a 3.19.(c) és a 3.19.(d) ábrán van. Például, ha négy bitet használunk a sorszámokhoz, azok 0 és 15 közé esnek (a határokat is beleértve). Ebben az esetben minden időpillanatban legfeljebb nyolc nyugtázatlan keretnek szabad kint lennie. Ha a vevő éppen elfogadta a 0.-tól 7.-ig tartó kereteket és léptette az ablakát, hogy lehetővé tegye a 8.-tól 15.-ig tartó keretek elfogadását, egyértelműen megmondható, hogy a következő keretek újraküldések (0.-tól 7.-ig), vagy újak (8.-tól 15.-ig). Általánosan, az ablakméret a 6. protokollnál $(MAX\_SEQ + 1)/2$.

Érdekes kérdés az, hogy a vevőben hány puffernek kell lennie. A vevő semmilyen körülmények között nem fogad el olyan kereteket, amelyek sorszáma kisebb, mint az ablak alsó széle, vagy nagyobb, mint a felső. Következésképp a pufferek száma az ablak méretével egyenlő, nem a sorszámok mennyiségével. A fenti, 4 bites sorszámokat tartalmazó példában nyolc – 0.-tól 7.-ig számozott – puffer kell. Amikor az *i.* keret megérkezik, az *i* mod 8. pufferbe kell tenni. Vegyük észre, hogy bár az *i.* és az $(i + 8)$ mod 8. ugyanazért a pufferért „verseng", soha nincsenek az ablakban egyszerre, mert ehhez legalább 9 hosszúságú ablak kellene.

Ugyanezért az időzítők száma is a pufferek számával egyenlő, nem pedig a sorszámmező méretével. Valójában minden pufferhez tartozik egy időzítő. Amikor az időzítő lejár, a puffer tartalmát újraküldjük.



3.19. ábra. *(a) Kezdeti állapot héthosszúságú ablakkal. (b) Hét keret elküldése és vétele után, de még a nyugtázás előtt. (c) Kezdeti állapot négyhosszúságú ablakkal. (d) Négy keret elküldése és vétele után, de még a nyugtázás előtt*

Az 5. protokollban van egy implicit feltételezés, miszerint a csatorna alaposan le van terhelve. Ha egy keret megérkezik, nem küldünk azonnal nyugtát, ráültetjük a következő kimenő adatkeretre. Ha a visszairányú forgalom kicsi, a nyugtát hosszú ideig várakoztatjuk. Ha nagy forgalom van az egyik irányban és nincs forgalom a másikban, csak *MAX_SEQ* darab keretet küldünk el, és ezután a protokoll blokkolódik.

A 6. protokollban ezt a problémát megoldottuk. Miután egy sorrendben következő adatkeret megérkezik, egy segédidőzítőt indítunk el a *start_ack_timer*-rel. Ha nem jelentkezik visszairányú forgalom, mielőtt az időzítő lejár, egy külön nyugtakeretet küldünk. A segéd időzítő által okozott megszakítás az *ack_timeout* esemény. Ezzel a megoldással most már egyirányú forgalom is lehetséges, mert a visszirányú adatkeretek (melyekre a nyugták ültethetők) hiánya már nem akadály. Csak egy segéd időzítő van, és ha a *start_ack_timer*-t meghívjuk, miközben az időzítő megy, az időzítő visszaáll a teljes időzítési intervallum elejére.

Lényeges, hogy a segéd időzítő időzítési intervalluma jelentősen rövidebb legyen a keretek időzítéséhez használt időzítőénél. Ez a feltétel ahhoz szükséges, hogy biztosak lehessünk benne, hogy egy rendben vett keret nyugtája megérkezik, mielőtt az adó időzítője lejárna és újraadná a keretet.

A 6. protokoll hatékonyabb stratégiát alkalmaz a hibák kezelésére, mint az 5. Amikor a vevőnek oka van gyanítani, hogy hiba történt, egy negatív nyugtakeretet (NAK) küld vissza az adó állomásnak. Egy ilyen keret kérés a NAK-ban megjelölt keret újraadására. Két eset van, amikor a vevő gyanakodhat: egy sérült vagy egy a várttól különböző keret érkezett (potenciális keretvesztés). Ahhoz, hogy elkerüljük, hogy ugyanarra az elveszett keretre több újraküldési igény jelentkezzen, a vevőnek nyomon kell követnie, hogy küldött-e már NAK-ot egy adott keretre. A *no_nak* változó értéke a 6. protokollban igaz, ha még nem küldtünk NAK-ot a *frame_expected*-re. Ha a NAK tönkremegy vagy elveszik, ez semmi kárt nem okoz, hiszen az adó állomásnak végül is lejár az időzítője, és mindenképpen újraküldi a keretet. Ha rossz keret érkezik, miután egy NAK-ot elküldtünk és az elveszett, a *no_nak* igaz lesz, és elindítjuk a segéd időzítőt. Amikor lejár, egy nyugtát küldünk, hogy újraszinkronizáljuk az adó állomást vevő aktuális állapotához.

Néhány helyzetben a keret célbajutásához, ottani feldolgozásához és a nyugta visszaéréséhez szükséges idő (közel) konstans. Ezekben a helyzetekben az adó állomás az időzítési intervallumát beállíthatja úgy, hogy éppen csak egy kicsit legyen hosszabb, mint az a várható idő, ami egy keret elküldése és nyugtájának vétele között telik el. Ha azonban ez az idő nagyon változó, az adónak azzal a döntéssel kell szembenéznie, hogy vagy kicsire állítja az intervallumot és szükségtelen újraküldéseket kockáztat, így sávszélességet pazarol el, vagy túl nagyra állítja és sokáig marad tétlen egy hiba után, így szintén pazarolja a sávszélességet. Ha a visszairányú forgalom szórványos, az az idő, ami előtt egy nyugta érkezése nem várható, rövidebb, amikor van visszirányú forgalom, és hosszabb, amikor nincs. A vevő változó feldolgozási ideje itt szintén probléma. Általában, amikor a nyugtázási időtartam szórása kicsi magához az időtartamhoz képest, az időzítést „szűk"-re lehet állítani, és a NAK-ok nem hasznosak. Egyébként az időzítést „tág"-ra kell állítani, és a NAK-ok észrevehetően felgyorsíthatják az elveszett vagy megsérült keretek újraküldését.

Szorosan kapcsolódva az időtúllépések és NAK-ok problémájához, felmerül azon

kérdés eldöntése, hogy melyik keret okozta az időtúllépést. Az 5. protokollban ez mindig az *ack_expected*, mert ez mindig a legidősebb. A 6. protokollban nincs triviális módja annak, hogy meghatározzuk, hogy melyik okozta az időtúllépést. Tegyük fel, hogy a 0.-tól a 4.-ig tartó kereteket küldtük el, ami azt jelenti, hogy a kint levő keretek listája 01234, a legidősebbtől a legfiatalabbig. Most képzeljük el, hogy a 0. időzítője lejár, az 5.-et (egy új keret) továbbítjuk, az 1. időzítője lejár, a 2. időzítője lejár, és a 6.-at (egy másik új keret) továbbítjuk. Ezen a ponton a kint levő keretek listája 3405126 a legidősebbtől a legfiatalabbig. Ha egy ideig minden bejövő forgalom elveszik, a hét kint levő keretnek ebben a sorrendben jár le az időzítője. Hogy ne bonyolítsuk tovább a példát, nem mutatjuk be az időzítők adminisztrációját. Ehelyett csak azt feltételezzük, hogy az *oldest_frame* változót időtúllépéskor beállítjuk, hogy jelezze, hogy melyik keret időzítése járt le.

3.5. Protokollok specifikációja és verifikációja

A valós protokollok, és az azokat megvalósító programok gyakran meglehetősen bonyolultak. Ezért sok kutatást végeztek, hogy formális matematikai módszereket találjanak a protokollok specifikációjához és verifikációjához. Ebben a fejezetben megnézünk néhány modellt és módszert. Bár ezeket az adatkapcsolati réteg kapcsán vizsgáljuk, alkalmazhatók más rétegekben is.

3.5.1. Véges állapotú automata modellek

A protokoll modellekben kulcsfogalom a **véges állapotú automata (finite state machine)**. Ebben a modellben mindegyik **protokoll automata (protocol machine)** – azaz az adó vagy a vevő – minden időpillanatban egy meghatározott állapotban van. Az automata állapota a változói értékéből áll, beleértve a programszámlálót is.

A legtöbb esetben elemzés céljából nagyszámú állapotot összevonhatunk. Például, vegyük a 3. protokollt. Az összes lehetséges állapotból két fontosat képezhetünk: a 0. keretre várakozást és az 1. keretre várakozást. Az összes többi állapotot vehetjük átmenetnek a két fő állapot között. Tipikusan az állapotokat úgy választjuk meg, hogy azokat a pillanatokat jellemezzék, amikor a protokoll a következő eseményre várakozik (azaz példáinkban a *wait_for_event* hívást hajtja végre). Ezen a ponton a protokoll automata állapotát teljesen meghatározzák változóinak állapotai. Az állapotok száma ekkor 2^n, ahol n az összes változó ábrázolásához szükséges bitek száma.

A teljes rendszer állapota a két protokoll automata állapotának és a csatorna állapotának a kombinációja. A csatorna állapotát annak tartalma határozza meg. Ismét a 3. protokollt használva példának: egy egyes vagy nullás keret halad az adótól a vevő felé, egy nyugtakeret megy a másik irányba, vagy üres a csatorna. Ha a küldőt és a vevőt két állapottal modellezzük, a teljes rendszernek 16 különböző állapota van.

A csatorna állapotáról beszélni teljesen jogos. Az a fogalom, hogy a keret „a csatornán" van, természetesen absztrakció. Amire itt gondolunk az az, hogy a keretet

részben továbbítottuk, részben vettük, de még nem dolgoztuk fel a célállomásnál. A keret „a csatornán", marad, amíg a protokoll automata végre nem hajtja a *from_physical_layer*-t, és fel nem dolgozza azt.

Minden állapotból nulla vagy több lehetséges **állapot-átmenet (transition)** vezet más állapotokba. Egy átmenet akkor következik be, amikor valamilyen esemény történik. A protokoll automatánál egy átmenet bekövetkezhet, ha elküldünk egy keretet, megérkezik egy keret, lejár egy időzítő, megszakítás érkezik stb. A csatornánál tipikus események: a protokoll automata egy új keretet helyez a csatornára, egy keret kézbesítődik egy protokoll automatához, vagy egy keret elveszik a zaj miatt. A protokoll automatáknak és a csatorna jellemzőinek teljes leírását megadva lehetővé válik, hogy rajzoljunk egy irányított gráfot, mely az állapotokat csomópontokként, az átmeneteket pedig irányított élekként tartalmazza.

Egy kitüntetett állapotot **kezdeti állapotnak (initial state)** jelölünk ki. Ez a rendszer indulásakor kialakuló állapot vagy valamilyen kényelmes kiinduló helyzet röviddel az elindulás után. A kezdeti állapotból néhány – lehetséges, hogy az összes – állapot elérhető átmenetek sorozatával. A gráfelmélet jól ismert módszereit alkalmazva (pl. a gráf tranzitív lezártját kiszámítva) meghatározható, hogy mely állapotok elérhetők, és melyek nem. Ezt a módszert **elérhetőség vizsgálatnak (reachability analysis)** nevezik (Lin és mások, 1987). Ez a vizsgálat hasznos lehet a protokoll helyességének megállapításában.

Formálisan egy protokoll véges állapotú automata modelljét egy négyesnek (S, M, I, T) tekinthetjük, ahol:

S a folyamatok és a csatorna lehetséges állapotainak halmaza.

M a csatornán továbbítható keretek halmaza.

I a folyamatok kezdeti állapotainak halmaza.

T az állapotok közötti átmenetek halmaza.

Kezdetben minden folyamat a kezdeti állapotában van. Ezután események történnek, például keretek kerülnek továbbításra vagy időzítők járnak le. Minden esemény kiválthatja az egyik folyamat vagy a csatorna valamilyen tevékenységét és új állapotba billenését. Gondosan felsorolva minden állapothoz tartozó összes lehetséges következő állapotot, felépíthetjük az elérhetőségi gráfot, és analizálhatjuk a protokollt.

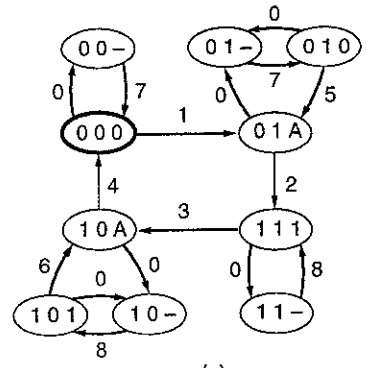
Az elérhetőségi vizsgálat a protokoll specifikáció különféle hibáinak felderítésére használható. Például, ha lehetséges, hogy egy bizonyos keret érkezik egy bizonyos állapotban, és a véges állapotú automata nem mondja meg, hogy mit kell tenni, a specifikáció hibás (nem teljes). Ha létezik állapotoknak olyan halmaza, amelyből nincs kijárat, és nem tudunk benne továbblépni (hibátlan kereteket venni), egy másik hibával állunk szemben (holtpont). A legkevésbé súlyosan hibás az olyan protokoll specifikáció, amely megadja, hogy hogyan kezeljünk egy eseményt egy olyan állapotban, amelyben az esemény nem következhet be (felesleges átmenet). Másfajta hibák szintén kimutathatók.

A véges állapotú automatára példaként tekintsük a 3.20.(a) ábrát. Ez a gráf a 3. protokollnak felel meg, ahogyan fentebb leírtuk: minden protokoll automatának két állapota van, a csatornának pedig négy. Összesen 16 állapot van, de nem mindegyik érhető el a kezdeti állapotból. Az elérhetetlen állapotokat nem tüntettük fel az ábrán. Minden állapotot három karakterrel – *XYZ* – címkéztünk meg, ahol *X* értéke 0 vagy 1 a küldő által küldeni próbált keretnek megfelelően; *Y* értéke szintén 0 vagy 1 a vevő által várt keretnek megfelelően, és *Z* értéke 0, 1 vagy üres (–), a csatorna állapotának megfelelően. Ebben a példában a kezdeti állapotnak a (000)-t választottuk. Más szóval a küldő éppen elküldte a 0. keretet, a vevő a 0. keretet várja és a 0. keret a csatornán van.

Kilencfajta átmenet látható a 3.20. ábrán. A 0. átmenet az, amikor a csatorna elveszti a tartalmát. Az 1. átmenet abból áll, hogy a csatorna helyesen továbbítja a 0. keretet a vevőhöz, a vevő átmegy az 1. keretet váró állapotba és kibocsát egy nyugtát. Az 1. átmenet további következménye még, hogy a vevő kézbesíti a hálózati rétegének a 0. keretet. A többi átmenetet a 3.20.(b) ábrán soroltuk fel. A hibás ellenőrző összegű keret érkezését nem tüntettük fel, mert nem változtatja meg az automata állapotát (a 3. protokollban).

A normális működés közben az 1., 2., 3. és 4. átmenetek ebben a sorrendben ismétlődnek újra és újra. Minden ciklusban két keretet kézbesítünk, és az adó visszajut a kezdeti állapotba, amelyben egy új 0-s sorszámú keret elküldésével próbálkozik. Ha a csatorna elveszti a 0. keretet, a (000)-ból a (00–) állapotba kerül az automata. Előbbutóbb az adó időzítője lejár (7. átmenet), és a rendszer visszatér a (000) állapotba. Egy nyugta elvesztése komplikáltabb: két átmenet (7. és 5. illetve 8. és 6.) szükséges a hiba kijavítására.

Az egyik tulajdonság, amivel egy 1 bites sorszámmal rendelkező protokollnak bírnia kell az az, hogy az eseményeknek mindegy, hogy milyen sorozata következik be, a vevő sohasem kézbesít két páratlan sorszámú csomagot egymás után egy páros sorszámú csomag nélkül, és viszont. A 3.20. ábra gráfjából látjuk, hogy ez a követelmény formálisabban úgy fogalmazható meg, hogy: „Nem létezhet olyan út a kezdeti állapot



(a)

Átmenet	Ki fut?	Fogadott keret	Kibocsátott keret	Hálózati rétegnek
0	–	(keret elveszett)		–
1	R	0	A	Igen
2	S	A	1	–
3	R	1	A	Igen
4	S	A	0	–
5	R	0	A	Nem
6	R	1	A	Nem
7	S	(időtúllépés)	0	–
8	S	(időtúllépés)	1	–

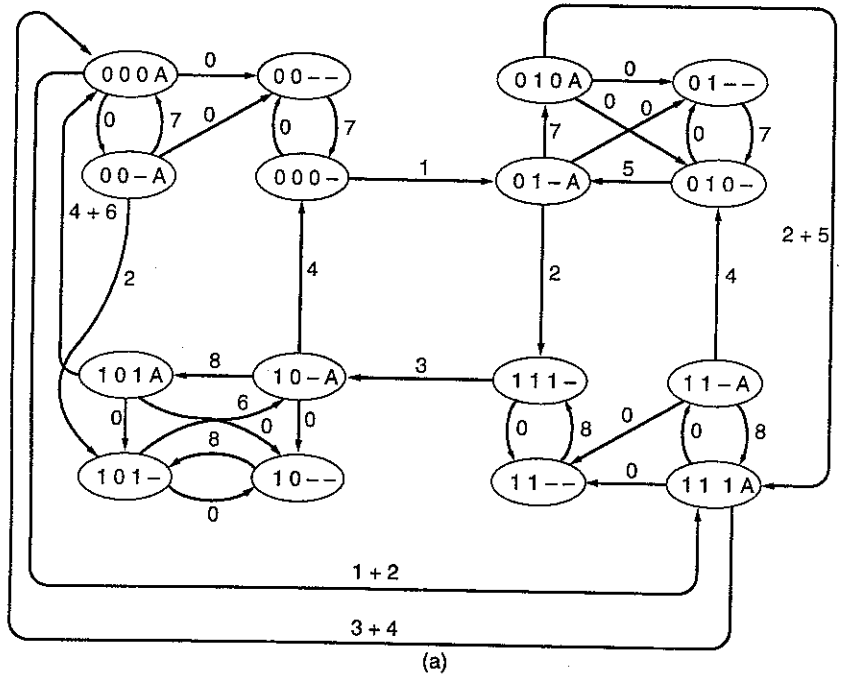
(b)

3.20. ábra. *(a) Állapot diagram a 3. protokollhoz. (b) Átmenetek*

ból kiindulva, amelyiken két 1. átmenet van anélkül, hogy közöttük 3. átmenet legyen, és viszont." Az ábrából látható, hogy a protokoll ebben a tekintetben korrekt.

Egy másik, hasonló követelmény az, hogy ne legyen olyan út, amelyen az adó kétszer vált állapotot (pl. 0-ból 1-be és vissza 0-ba), míg a vevő állapota nem változik. Ha egy ilyen út létezne, az ennek megfelelő eseménysorozatban két keret menthetetlenül elveszne anélkül, hogy a vevő észrevette volna. Így a kézbesített csomag sorozatban lenne egy észrevétlen, kétcsomagnyi hiány.

Még egy fontos tulajdonsága a protokollnak, a holtpontmentesség. A **holtpont (deadlock)** olyan szituáció, amelyben a protokoll nem tud továbblépni (pl. csomagokat kézbesíteni a hálózati rétegnek) függetlenül attól, hogy milyen eseménysorozat történik. A gráf modell szempontjából a holtpont az állapotok egy olyan részhalmazának létezésével jellemezhető, amely elérhető a kezdeti állapotból és két jellemzővel bír:



(a)

(0 0 0 –), (0 1 – A), (0 1 0 A), (1 1 1 A), (1 1 – A), (0 1 0 –), (0 1 – A) (1 1 1 –)

(b)

3.21. ábra. *(a) Állapotgráf a 3. protokollhoz duplex csatornával. (b) A protokollhibát okozó állapotsorozat*

1. Nincs a részhalmazból kivezető átmenet.

2. Nincs olyan átmenet a részhalmazban, amely továbblépést eredményez.

Ha a protokoll egyszer bekerül egy holtpont helyzetbe, örökre ott is marad. Megint csak könnyen látható a gráfból, hogy a 3. protokoll nem küzd holtpont problémával.

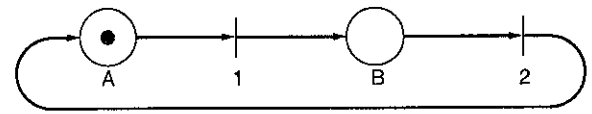
Most vegyük a 3. protokollnak egy olyan változatát, amelyben a félduplex csatornát duplexre cseréljük. A 3.21. ábrán látható állapotok a két protokoll automata és a két csatorna állapotainak Descartes-szorzataként adódik. Vegyük észre, hogy az előreirányú csatornának most három állapota van: 0. keret, 1. keret vagy üres, visszairányú csatornának pedig kettő: *A* vagy üres. Az átmenetek ugyanazok, mint a 3.20.(b) ábrán, kivéve azt a kis különbséget, ami arra az esetre vonatkozik, amikor egy nyugta- és egy adatkeret egyszerre van a csatornán. A vevő nem veheti el az adatkeretet, mert az maga után vonná, hogy két nyugta legyen a vonalon egyszerre, ami a modellünkben nem megengedett (bár könnyen kitalálható egy olyan modell, amely ezt megengedi). Hasonlóan, az adó sem veheti el a nyugtát, mert az maga után vonná egy második adatkeret kibocsátását, mielőtt az elsőt a vevő elfogadta volna. Következésképp a két eseménynek egyszerre kell bekövetkeznie. Ennek megfelelően például a (000A) és a (111A) állapot közötti átmenetet 1 + 2-vel jelöltük az ábrán.

A 3.21. ábrán vannak olyan utak, amelyek a protokoll hibás működését okozzák. Konkrétabban olyanok, melyek mentén az adó újra és újra lekér csomagokat annak ellenére, hogy az előzőeket nem kézbesítette rendben. A probléma azért jelentkezik, mert itt lehetséges az, hogy az adó állomásnak lejárjon az időzítője és új keretet küldjön anélkül, hogy megzavarná a nyugtát a visszairányú csatornán. Amikor a nyugta megérkezik, tévesen a jelenlegi, és nem az előző átvitelre vonatkozónak tekinthetjük.

Egy protokollhibát okozó állapotsorozat látható a 3.21.(b) ábrán. A sorozat negyedik és hatodik állapotában az adó állapotot vált, jelezve, hogy új csomagot kér le a hálózati rétegtől, miközben a vevő nem vált állapotot, azaz nem kézbesít csomagot a hálózati rétegnek.

3.5.2. Petri-gráf modellek

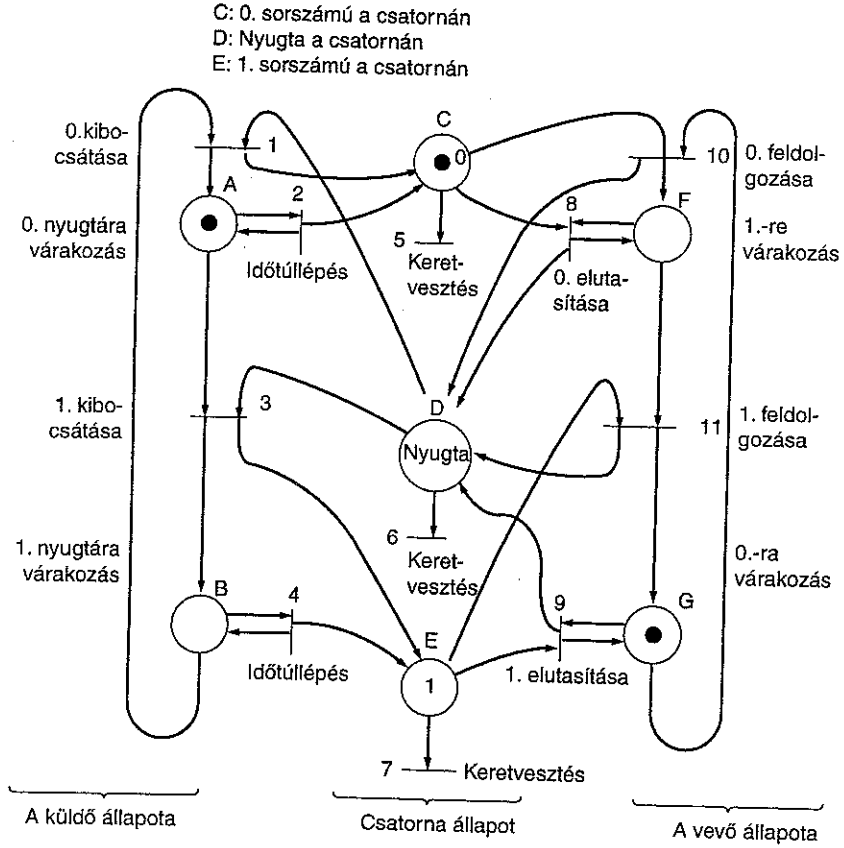
A véges állapotú automata nem az egyetlen módszer a protokollok formális specifikálására. Ebben a szakaszban egy másik módszert, a **Petri-gráfot (Petri Net** – Danthine, 1980) mutatjuk be. Egy Petri-gráfnak négy alapeleme van: helyek, átmenetek, élek és tokenek. A **hely (place)** egy állapotot reprezentál, amelyben a rendszer (vagy annak egy része) lehet. A 3.22. ábra egy Petri-gráfot mutat két hellyel – *A*-val és



3.22. ábra. *Egy Petri-gráf két hellyel és két átmenettel*

B-vel –, mindkettőt körrel jelölve. A rendszer jelenleg az *A* állapotban van, ezt jelzi a **token (vastag pont)** az *A* helyen. Az **átmenetet** egy vízszintes vagy függőleges vonal szimbolizálja. Minden átmenetnek nulla vagy több **bemeneti éle (input arc)** van, melyek a bemeneti helyeikből jönnek, és nulla vagy több **kimeneti éle (output arc)** van, melyek a kimeneti helyeik felé mennek.

Egy átmenet **engedélyezett (enabled),** ha minden egyes bementi helyén legalább egy token van. Bármelyik engedélyezett átmenet **tüzelhet (fire),** ha akar, eltávolítva mindegyik bemeneti helyéről egy tokent, és elhelyezve egyet mindegyik kimeneti helyén. Ha a bemeneti és a kimeneti élek száma különbözik, a tokenek száma megváltozik. Ha kettő vagy több átmenet engedélyezett, bármelyikük tüzelhet. Annak a kiválasztása, hogy melyik átmenet tüzeljen, meghatározatlan, éppen ezért hasznos a Petri-gráf a protokollmodellezéshez. A 3.22. ábra Petri-gráfja determinisztikus és bármely kétfázisú folyamat modellezésére használható (pl. egy csecsemő viselkedésére: eszik, alszik, eszik, alszik és így tovább). Mint minden modellezési eszköznél, a szükségtelen részleteket elhagytuk.



3.23. ábra. *A 3. protokoll Petri-gráf modellje*

A 3.23. ábrán megadjuk a 3.21. ábra Petri-gráf modelljét. Ellentétben a véges állapotú automata modellel, itt nincsenek összetett állapotok; az adó állapotát, a csatorna-állapotot és a vevő állapotát külön-külön jelenítjük meg. Az 1. és a 2. átmenet a 0. keret normális, illetve időtúllépéskori továbbításának felel meg. A 3. és 4. átmenetek analógak az 1. keretre vonatkozóan. Az 5., 6. és 7. átmenet a 0. keret, a nyugta-, illetve az 1. keret elvesztésének felel meg. A 8. és 9. átmenet akkor következik be, amikor egy rossz sorszámmal rendelkező keret érkezik a vevőhöz. A 10. és 11. átmenet a sorrendben következő keret vevőhöz való megérkezését, és a hálózati réteghez való kézbesítését reprezentálja.

A Petri-gráfok hasonló módon használhatók protokollhibák felfedésére, mint a véges állapotú automaták. Például, ha valamilyen tüzelési sorozat kétszer tartalmazná a 10. átmenetet anélkül, hogy közöttük a 11.-et is tartalmazná, a protokoll hibás lenne. A holtpont fogalma a Petri-gráfban szintén hasonló ahhoz, mint a véges állapotú automata megfelelőjében.

A Petri-gráfok kényelmes, nyelvtanhoz hasonló algebrai formában ábrázolhatók. Mindegyik átmenet a nyelvtan egy szabályát adja. A szabályok az átmenet bemeneti és kimeneti helyeit írják le. Például a 3.23. ábra 1. átmenete $BD \rightarrow AC$. A Petri-gráf aktuális állapotát a helyek rendezetlen gyűjteményével reprezentálhatjuk, melyben mindegyik hely annyiszor szerepel, ahány tokent tartalmaz. Bármelyik szabály, melynek mindegyik bal oldali helye benne van az állapotban, tüzelhet, eltávolítva a bal oldali helyeit, és hozzáadva a kimeneti helyeit. A 3.23. ábra jelölése *ACG*, így a 10. szabály ($CG \rightarrow DF$) használható, de a 3. ($AD \rightarrow BE$) nem.

3.6. Példák adatkapcsolati protokollokra

A következő szakaszokban megvizsgálunk számos széleskörűen alkalmazott protokollt. Az első – a HDLC – általános az X.25-ös és sok más hálózatban. Ezután megvizsgáljuk az Internetben, illetve az ATM hálózatokban használt protokollokat. A későbbi fejezetekben is elővesszük az Internetet és az ATM-et, mint élő példát.

3.6.1. HDLC – magas szintű adatkapcsolat-vezérlés (High-level Data Link Control)

Ebben a szakaszban a protokollok egy közeli rokonságban levő csoportját vizsgáljuk meg, melyek egy kicsit öregek ugyan, de még mindig sűrűn alkalmazzák a hálózatokban szerte a világon. Ezek mind az IBM SNA-jában alkalmazott adatkapcsolati protokollból származnak, melyet **SDLC (Synchronous Data Link Control – szinkron adatkapcsolat-vezérlés)** protokollnak neveznek. Az IBM miután kifejlesztette az SDLC-t, benyújtotta az ANSI-hoz és az ISO-hoz, hogy fogadják el egyesült államokbeli, illetve nemzetközi szabványnak. Az ANSI **ADCCP-re (Advanced Data Com-**

munication Control Procedure – fejlett adatkommunikáció-vezérlő eljárás), az ISO pedig **HDLC-re (High-level Data Link Control – magas szintű adatkapcsolat-vezérlés)** módosította. A CCITT ezután átvette és módosította a HDLC-t a saját **LAP-jévé (Link Access Procedure – adatkapcsolati elérési eljárás),** ami az X.25-ös hálózati interfész szabvány része, de később újra módosította **LAPB-vé,** hogy kompatíbilisebbé tegye a HDLC egy későbbi verziójával. A szabványokban az a szép, hogy olyan sok közül lehet válogatni. Ráadásul, ha egyik sem tetszik, csak meg kell várni a következő évi modellt.

Ezek a protokollok mind ugyanazon az elven alapulnak. Mindegyik bitalapú, és bitbeszúrást alkalmaz a kódfüggetlenség érdekében. Csak kisebb, ám mégis idegesítő részletekben különböznek. A bitalapú protokollok tárgyalását a következőkben általános bevezetőnek szánjuk. A protokollok sajátos részleteire vonatkozóan a megfelelő protokoll definíciót érdemes forgatni.

Mindegyik bitalapú protokoll a 3.24. ábrán látható keretstruktúrát használja. A *Cím* mező elsősorban a több terminállal rendelkező vonalaknál érdekes, ahol egy terminál azonosítására használatos. A két pontot összekötő vonalaknál néha a parancsok és a válaszok megkülönböztetésére használják.

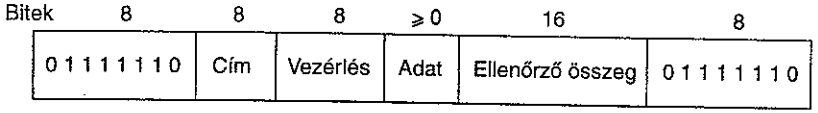
A *Vezérlés* mező sorszámozásra, nyugtázásra és egyéb célokra használatos, az alább leírtaknak megfelelően.

Az *Adat* mező tetszőleges információt tartalmazhat, és tetszőleges hosszúságú lehet, bár az ellenőrző összeg hatékonysága a kerethossz növelésével romlik a többszörös csoportos hibák nagyobb valószínűsége miatt.

Az *Ellenőrző összeg* mező a jól ismert ciklikus redundanciakód egy kisebb változtatással, generátor polinomként a CRC-CCITT-t alkalmazva. A változtatás lehetővé teszi, hogy észleljük az elveszett jelző bájtokat.

A keretet egy jelző sorozat (01111110) határolja. A tétlen kétpontos vonalakon folyamatosan továbbítjuk a jelző sorozatokat. A minimális keret három mezőt, összesen 32 bitet tartalmaz a két végén levő jelző sorozatokat nem számítva.

Háromféle keret van: **információs (Information), felügyelő (Supervisory)** és **számozatlan (Unnumbered).** A *Vezérlés* mező tartalma ezekhez a fajtákhoz a 3.25. ábrán látható. A protokoll csúszóablakot alkalmaz 3 bites sorszámmal. Legfeljebb hét nyugtázatlan keret lehet kint bármely pillanatban. A *Sorszám* mező a 3.25.(a) ábrán a keret sorszáma. A *Következő* mező a ráülletett nyugta. Mindegyik protokoll ahhoz a konvencióhoz tartja magát, hogy az utolsó hibátlanul vett keret sorszáma helyett az első nem vett keret sorszámát (azaz a következő várt keretét) ülteti rá a visszairányú adatra. Annak megválasztása, hogy az utolsó vett keretet vagy az első nem vett keretet használjuk tetszőleges; mindegy, hogy melyik konvencióhoz ragaszkodunk, ha azt következetesen tesszük.



3.24. ábra. *Keret formátum a bitalapú protokollokhoz*

A *L/U* bit a *Lekérdezés/Utolsó (Poll/Final)* rövidítése. Ezt akkor használjuk, amikor egy számítógép (vagy koncentrátor) terminálok egy csoportját kérdezi le. Amikor $L = 1$, a számítógép felkéri a terminált adatküldésre. A terminál által küldött mindegyik keretben a *L/U* bit $U = 0$-ra van állítva, kivéve az utolsót, ami $U = 1$.

Némelyik protokollban a *L/U* bitet arra használjuk, hogy kényszerítsük a másik gépet egy felügyelő keret azonnali elküldésére, ahelyett, hogy a visszirányú forgalmat megvárva arra ültesse az ablak információt. A bitnek a számozatlan keretekkel kapcsolatosan is van egy kevésbé jelentős szerepe.

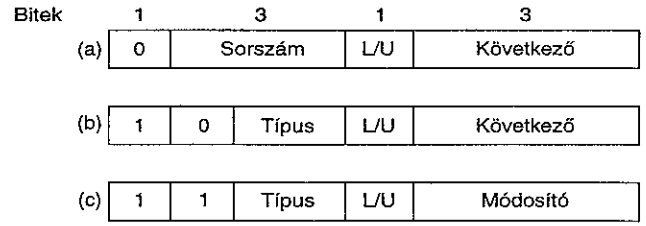
A különféle felügyelő kereteket a *Típus* mező alapján lehet megkülönböztetni. A 0. típus a nyugtakeret (hivatalosan VÉTELRE KÉSZ-nek – RECEIVE READY – nevezik), mely arra használatos, hogy jelezze a következő várt keretet. Ezt a keretet akkor használjuk, ha nincs ráültetésre használható visszairányú forgalom.

Az 1. típus egy negatív nyugtakeret (hivatalosan ELUTASÍTÁS-nak – REJECT – nevezik). Ezt átviteli hiba jelzésére használjuk. A *Következő* mező jelzi az első keretet a sorban, amelyiket nem vettünk hibátlanul (azaz az újraküldendő keretet). Az adónak újra kell küldenie az összes kint levő keretet a *Következő* mezőben szereplő sorszámtól kezdődően. Ez a stratégia leginkább az 5. protokollunkban alkalmazott megoldáshoz hasonlít.

A 2. típus a VÉTELRE NEM KÉSZ (RECEIVE NOT READY). Ez nyugtáz minden keretet a *Következő*-ig (de a *Következő*-t már nem beleértve), ugyanúgy, mint a VÉTELRE KÉSZ, de ez tudatja az adóval, hogy állítsa le a keretek küldését. A VÉTELRE NEM KÉSZ jelzést arra szánták, hogy átmeneti problémákat jelezzen, mint amilyen pl. a szabad puffer hiánya, nem pedig, hogy a csúszóablakos forgalomszabályozás alternatívája legyen. Amikor a feltételek helyreálltak, a vevő VÉTELRE KÉSZ, ELUTASÍTÁS vagy más vezérlő keretet küld.

A 3. típus a SZELEKTÍV ELUTASÍTÁS (SELECTIVE REJECT). Ez csak megadott keretek újraküldésére szólít fel. Ebben az értelemben leginkább a 6. protokollunkra hasonlít, ezért ott hasznos, ahol a küldő ablakának mérete fele vagy kisebb, mint a sorszámmezőé. Így, ha a vevő pufferelni szeretné a sorrenden kívüli kereteket a lehetséges jövőbeni felhasználás érdekében, kikényszerítheti bármelyik adott keret újraküldését a SZELEKTÍV ELUTASÍTÁS segítségével. A HDLC és az ADCCP megengedi ezt a keret típust, de az SDLC és a LAPB nem (azaz nincs SZELEKTÍV ELUTASÍTÁS).

A keretek harmadik osztályát a számozatlan keretek alkotják. Ezek néha vezérlési célokra használatosak, de alkalmazhatók adatok átvitelére is, amikor megbízhatatlan összeköttetés nélküli szolgálatra van igény. A különböző bitalapú protokollok, ellentétben az előző két keretfajtával, ahol majdnem azonosak, ezen a ponton jelentősen



3.25. ábra. *(a) Egy információs, (b) egy felügyelő és (c) egy számozatlan keret vezérlő mezője*

különböznek. Öt bit áll rendelkezésre a kerettípus jelzésére, de nem használják mind a 32 lehetőséget.

Mindegyik protokoll tartalmazza a DISC (DISConnect – Kapcsolatbontás) parancsot, amely lehetővé teszi, hogy egy gép bejelentse, hogy le fog kapcsolódni (pl. megelőző karbantartás céljából). Van egy olyan parancs is, amely lehetővé teszi egy gépnek, amely éppen most kapcsolódott vissza a vonalra, hogy bejelentse a jelenlétét, és az összes sorszámot visszaállíttassa nullára. Ezt a parancsot SNRM-nek (Set Normal Response Mode – normál válaszmód beállítása) nevezik. Sajnos a „normál válaszmód" minden, csak nem normális. Ez egy kiegyensúlyozatlan (aszimmetrikus) mód, amiben a vonal egyik vége a mester (master) a másik a szolga (slave). Az SNRM abból az időből származik, amikor az adatkommunikáció azt jelentette, hogy egy buta terminál beszélget egy okos számítógéppel, amin tisztán látszik, hogy aszimmetrikus. Hogy a protokoll alkalmasabb legyen olyan esetekre, amikor a két partner egyenrangú, a HDLC-ben és a LAPB-ben van egy további parancs, az SABM (Set Asynchronous Balanced Mode – aszinkron kiegyensúlyozott mód beállítása), amely alaphelyzetbe állítja a vonalat, és a két partnert egyenrangúnak nyilvánítja. A fenti két protokollnak van még két további parancsa is, az SABME és az SNRME, melyek ugyanazok, mint az SABM, illetve az SNRM, kivéve, hogy engedélyezik a kiterjesztett keret formátumot, amelyben 7 bites sorszámok használhatók a 3 bitesek helyett.

Egy harmadik parancs, amit mindegyik protokoll tartalmaz az FRMR (FRaMeReject – keretelutasítás), amivel egy helyes ellenőrző összegű, de lehetetlen szemantikájú keret érkezését jelezzük. Lehetetlen szemantikájú keret például egy 3. típusú felügyelő keret a LAPB-ben, egy 32 bitnél rövidebb keret, egy illegális vezérlőkeret vagy egy olyan keretre érkező nyugta, ami kívül esik az ablakon stb. Az FRMR keretek tartalmaznak egy 24 bites adat mezőt, amely azt mutatja, hogy mi volt rossz a kerettel kapcsolatban. Az adat tartalmazza a rossz keret vezérlő mezőjét, az ablak paramétereket, és egy bitsorozatot, amely a konkrét hiba jelzésére használatos.

A vezérlő keretek ugyanúgy elveszhetnek vagy megsérülhetnek, mint az adatkeretek, ezért nyugtázni kell őket is. Egy speciális vezérlő keret biztosítja ezt, melyet UA-nak (Unnumbered Acknowledgement – számozatlan nyugta) neveznek. Mivel csak egy vezérlő keret lehet kinn, mindig egyértelmű, hogy melyik vezérlő keretet nyugtázzuk.

A többi vezérlő keret az inicializációval, lekérdezéssel és az állapotjelentéssel kapcsolatos. Van egy olyan vezérlő keret is, amely tetszőleges információt tartalmazhat. Ezt UI-nak (Unnumbered Information – számozatlan adatok) nevezik. Az ebben levő adatot nem továbbítjuk a hálózati rétegnek, az a vevő adatkapcsolati rétegnek szól.

Széles körű alkalmazása ellenére a HDLC messze nem tökéletes. A vele kapcsolatos változatos problémák tárgyalása megtalálható (Fiorini és mások, 1995) műveiben.

3.6.2. Az Internet adatkapcsolati rétege

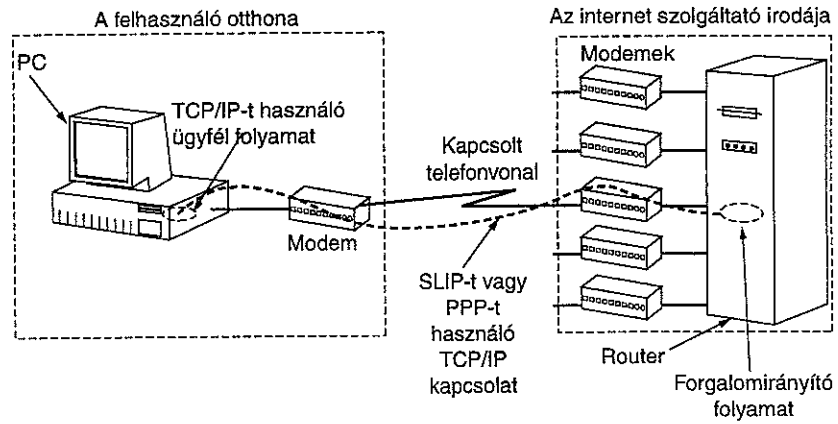
Az Internet önálló gépekből (hosztokból és routerekből), és az őket összekapcsoló kommunikációs infrastruktúrából áll. Egy épületen belül többnyire LAN-okat alkalmaznak az összekapcsolásra, de a legtöbb nagy kiterjedésű infrastruktúra kétpontos

összeköttetéseket megvalósító bérelt vonalakból épül fel. A 4. fejezetben a LAN-okat fogjuk megnézni; itt az Internetben alkalmazott kétpontos vonalak adatkapcsolati protokolljait fogjuk megvizsgálni.

A gyakorlatban a kétpontos kommunikációt elsősorban két helyzetben alkalmazzák. Az egyik: szervezetek ezreinek van egy vagy több LAN-ja néhány hoszttal (személyi számítógépek, felhasználói munkaállomások, kiszolgálók és így tovább) és a LAN-on egy routerrel (vagy híddal, ami hasonló funkciójú). Gyakran a routereket összekötik egy gerinchálózattal. Tipikusan minden kapcsolat a külvilággal egy vagy két routeren keresztül megy, melyeknek kétpontos bérelt vonali kapcsolatuk van távoli routerekkel. Ezek a routerek és bérelt vonalaik alkotják azt a kommunikációs alhálózatot, amelyre az Internet épül.

A másik szituáció, ahol a kétpontos vonalak főszerepet játszanak, az otthoni, modemes, kapcsolt telefonvonalas eléréssel rendelkező egyéni felhasználók millióinak összekapcsolása az Internettel. Rendszerint ez úgy történik, hogy a felhasználó otthoni PC-je felhív egy **Internet-szolgáltatót (Internet provider)**, mely lehet kereskedelmi cég, mint az America Online, CompuServe vagy a Microsoft Network, de sok egyetem és cég szintén biztosít otthoni Internet-kapcsolatot hallgatói és dolgozói számára. Néha az otthoni PC csak karakteres terminálként működik, mellyel be lehet jelentkezni az Internet-szolgáltató időosztásos rendszerébe. Így a felhasználó parancsokat írhat be, programokat futtathat, de a grafikus Internet-szolgáltatások, mint például a World Wide Web, nem érhetők el. A munkának ezt a módját **felületi hozzáférésnek (shell account)** nevezik.

Egy másik megoldás, ha az otthoni PC felhívhatja az Internet-szolgáltató routerét, és ezután egy teljes funkcionalitással rendelkező Internet-hosztként működhet. Ez a módszer nem különbözik attól, mintha bérelt vonalunk lenne a PC és a router között, kivéve, hogy amikor a felhasználói viszony lezárul, a kapcsolat megszakad. Ezzel a megközelítéssel minden Internet-szolgáltatás – még a grafikusak is – elérhetővé válik. A 3.26. ábrán illusztráltuk, amint egy otthoni PC felhív egy Internet-szolgáltatót.



3.26. ábra. *Egy otthoni számítógép Internet-hosztként működtetve*

Mind a két router közötti bérelt vonalas, mind a hoszt és a router közötti kapcsolt vonalas összeköttetéshez szükség van a vonalon valamilyen adatkapcsolati protokollra a keretezés, hibavédelem és a többi általunk tanulmányozott adatkapcsolati funkció biztosításához. Két ilyen protokollt alkalmaznak széleskörűen az Internetben: az SLIP-t, és a PPP-t. Most meg fogjuk ezeket vizsgálni sorjában.

SLIP – (Serial Line IP) Soros vonali IP

Az **SLIP** az idősebb a két protokoll közül. Rick Adams találta ki 1984-ben ahhoz, hogy Sun munkaállomásokat tudjon csatlakoztatni az ARPANET-hez kapcsolt vonalon keresztül, modem segítségével. A protokoll, mely az RFC 1055-ben található, nagyon egyszerű. A munkaállomás nyers IP csomagokat küld át a vonalon egy speciális jelző bájttal (0xC0) a keret végén. Ha a jelző bájt előfordul az IP csomag belsejében, egyfajta karakterbeszúrást alkalmazunk: egy kétbájtos sorozatot (0xDB, 0xDC) küldünk helyette. Ha a 0xDB előfordul a csomagban, szintén beszúrást alkalmazunk. Néhány SLIP megvalósítás az elküldött IP csomag elejére és végére is csatol jelző bájtot.

Az SLIP újabb verziói valamiféle TCP és IP fejléctömörítést valósítanak meg. Ez annak a ténynek a kihasználásán alapul, hogy az egymást követő csomagok fejlécében sok közös mező van. Ezeket úgy tömörítjük, hogy kihagyjuk azokat a mezőket, amelyek azonosak az előző IP csomag megfelelő mezőivel. Ezen kívül azokat a mezőket sem küldjük el teljes egészében, amelyek különbözőek, csak a növekményüket az előző értékhez képest. Ezek az optimalizációk az RFC 1144-ben találhatók.

Bár az SLIP még mindig széleskörűen használatos, van néhány komoly probléma vele. Először is, nem csinál semmilyen hibajelzést vagy javítást, így a felsőbb rétegekre marad, hogy észleljék és helyreállítsák az elveszett, sérült vagy összeolvadt keretekből származó hibákat.

Másodszor, az SLIP csak az IP-t támogatja. A növekvő Internet egyre több olyan hálózatot foglal magába, aminek nem anyanyelve az IP (pl. a Novell LAN-ok), így ez a korlátozás egyre komolyabb problémává válik.

Harmadszor, mindkét oldalnak előre kell tudnia a másik IP címét; egyik címet sem lehet a kapcsolat felépítése során dinamikus hozzárendeléssel megállapítani. Az IP címek számának jelenlegi szűkössége mellett ez a korlátozás nagyon fontos szempont, hiszen egyszerűen lehetetlen mindegyik otthoni Internet-felhasználónak egyedi IP címet adni.

Negyedszer, az SLIP nem biztosít semmiféle hitelesítést, így egyik fél sem tudja, hogy kivel beszélget valójában. A bérelt vonalaknál ez nem igazán szempont, de a kapcsolt vonalaknál az.

Ötödször, az SLIP nem jóváhagyott Internet-szabvány, így sok különböző (és inkompatíbilis) változata létezik. Ez nem teszi egyszerűvé az együttműködést.

PPP – Pont-pont protokoll (Point-to-Point Protocol)

Hogy javítson a helyzeten az IETF, felkért egy csoportot, hogy találjanak ki egy olyan protokollt a kétpontos vonalakhoz, amelyik az összes problémát megoldja, és hivatalos Internet-szabvánnyá válhat. Ennek a munkának az eredménye a **PPP (pont-pont protokoll – Point-to-Point Protocol),** melyet az RFC 1661-ben definiáltak, és később tovább részletezték számos más RFC-ben (pl. az 1662. és 1663.). A PPP megoldja a hibajelzést, többféle protokollt támogat, lehetővé teszi, hogy az IP címeket a kapcsolat felépítésekor rendeljék az állomásokhoz, lehetővé teszi a hitelesítést, és még sok más fejlesztést tartalmaz az SLIP-hez képest. Bár még sok Internet-szolgáltató támogatja az SLIP-t és a PPP-t is, tisztán látszik, hogy a jövő a PPP, nemcsak a kapcsolt vonalak számára, hanem a két router közötti bérelt vonalak számára is.

A PPP három dolgot biztosít:

1. Olyan keretezési módszert, amely egyértelműen ábrázolja a keret végét és a következő keret kezdetét. A keretformátum megoldja a hibajelzést is.

2. Kapcsolatvezérlő protokollt a vonalak felélesztésére, tesztelésére, az opciók megbeszélésére és a vonalak elegáns elengedésére, amikor már nincs rájuk szükség. Ezt a protokollt **LCP-nek (adatkapcsolat-vezérlő protokoll – Link Control Protocol)** nevezik.

3. Olyan módot a hálózatiréteg-opciók megbeszélésére, amely független az alkalmazott hálózatiréteg-protokolltól. A választott módszer az, hogy különböző **NCP (hálózati vezérlő protokoll – Network Control Protocol)** van mindegyik támogatott hálózati réteghez.

Hogy lássuk, hogyan illenek össze ezek a darabok, vegyünk egy tipikus eseménysorozatot: az otthoni felhasználó felhív egy Internet-szolgáltatót, hogy az otthoni PC-je átmenetileg Internet-hoszttá válhasson. A PC először felhívja a szolgáltató routerét modemen keresztül. Miután a routerrel létrejött a fizikai kapcsolat, a PC LCP csomagok sorozatát küldi el egy vagy több PPP keret adat mezőjében. Ezek a csomagok és a rájuk adott válaszok választják ki az alkalmazandó PPP paramétereket.

Amikor ezekben megegyeztek, NCP csomagok sorozatát küldik el, hogy felkonfigurálják a hálózati réteget. A PC tipikusan TCP/IP protokoll készletet akar futtatni, ezért szüksége van IP címre. Nincs azonban elég IP cím ahhoz, hogy mindenkinek jusson, így rendszerint az Internet-szolgáltató kap valamennyi címet, és a bejelentkezési viszony idejére dinamikusan rendeli hozzá minden becsatlakozó PC-hez. Ha a szolgáltatónak *n* darab IP cím van a birtokában, legfeljebb *n* állomás lehet egyidejűleg bejelentkezve, de a teljes ügyfélbázisa ennek sokszorosa lehet. Az IP-hez készült NCP-t használjuk az IP címhozzárendeléshez.

Ezen a ponton a PC egy Internet-hoszt, és ugyanúgy küldhet és fogadhat IP csomagokat, mint egy a hálózattal állandó kapcsolatban levő hoszt. Amikor a felhasználó végzett, az NCP-t használjuk a hálózati rétegek kapcsolatának lebontására, és az IP cím felszabadítására. Ezután az LCP-vel zárjuk le az adatkapcsolati rétegbeli össze-

köttetést. Végül a számítógép szól a modemnek, hogy bontsa a vonalat, megszüntetve ezzel a fizikai rétegbeli kapcsolatot.

A PPP keretformátumát úgy választották meg, hogy minél jobban hasonlítson a HDLC keretformátumára. (Minek újra feltalálni a kereket?) A legfőbb különbség a PPP és a HDLC között az, hogy az előbbi karakter-, nem pedig bitalapú. A PPP az SLIP-hez hasonlóan karakterbeszúrást alkalmaz a kapcsolt modemvonalakon, így minden keretben egész számú bájt található. A HDLC-vel ellentétben, nem lehet olyan keretet küldeni amiben 30,25 darab bájt van. A PPP kereteket nem csak kapcsolt telefonvonalakon lehet elküldeni, hanem SONET-en vagy igazi bit alapú HDLC vonalakon is (pl. router-router kapcsolatokhoz). A PPP keretformátum a 3.27. ábrán látható.

Minden PPP keret a szabványos jelző bájttal (01111110) kezdődik, és karakterbeszúrást alkalmazunk, ha ez előfordul az adat mezőben. A következő a *Cím* mező, amelyik mindig a bináris 11111111 értékre van állítva, hogy jelezze, hogy minden állomásnak vennie kell a keretet. Ennek az értéknek a használatával elkerülhetjük az adatkapcsolati címek hozzárendelésének problémáját.

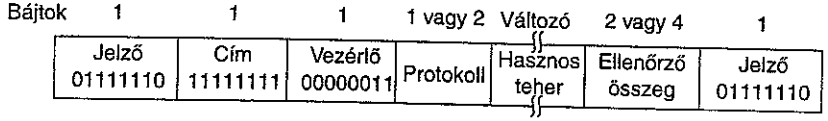
A *Cím* mezőt a *Vezérlő* mező követi, melynek alapértéke 00000011. Ez az érték a számozatlan keretet jelzi. Alapértelmezésben a PPP nem biztosít megbízható átvitelt (nem alkalmaz sorszámokat és nyugtákat). Zajos környezetben (mint például a vezeték nélküli hálózatok) a megbízható átvitel a számozott mód használatával megoldható. A pontos részletek az RFC 1663.-ban találhatóak.

Mivel a *Cím* és a *Vezérlő* mezők mindig ugyanazok az alapbeállításban, az LCP biztosítja a szükséges mechanizmust a két résztvevő számára, hogy kihagyják ezeket a mezőket, és így megtakarítsanak 2 bájtot keretenként.

A negyedik mező a *Protokoll* mező. Ennek a feladata az, hogy megmutassa, hogy milyen csomag van az adat mezőben. Az LCP, NCP, IP, IPX, AppleTalk és más protokollok számára kódok vannak definiálva. A 0-s bittel kezdődő protokollok hálózatiréteg-protokollok, mint például az IP, IPX, OSI CLNP, XNS. Az 1-es bittel kezdődően más, további egyeztetést igénylő protokollok lehetnek. Ezek között van az LCP és különböző NCP-k minden egyes támogatott hálózatiréteg-protokollhoz. A *Protokoll* mező hosszának alapértéke 2, de ez levihető 1-re az LCP segítségével történő megegyezés alapján.

Az *Adat* mező változó hosszúságú, de legfeljebb a megegyezett maximumnak megfelelő méretű. Ha a hosszban nem egyeznek meg a felek a vonal beállítása alatt az LCP segítségével, az alapértelmezés 1500 bájt. Ha szükség van rá, az adat mezőt kitöltő bájtok követhetik.

Az *Adat* mező után az *Ellenőrző összeg* jön, amely rendszerint 2 bájtos, de a felek megegyezhetnek 4 bájtos ellenőrző összegben is.



3.27. ábra. *A PPP teljes keret formátuma a számozatlan módú működéshez*

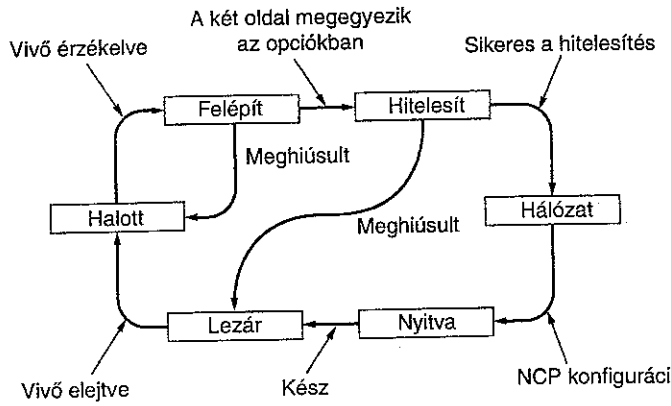
Összefoglalva: PPP egy többprotokollos keretezési eljárás, amely alkalmas modem, HDLC bit-soros vonal, SONET és más fizikai rétegek feletti használatra. Támogatja a hibajelzést, az opciókban való megegyezést, a fejléctömörítést és opcionálisan a megbízható átvitelt HDLC keretezéssel.

Most pedig a PPP keretformátumáról fordítsuk figyelmünket arra, hogy hogyan épül fel, illetve bomlik le egy kapcsolat. A 3.28. ábra leegyszerűsítve mutatja, hogy milyen fázisokon megy keresztül egy kapcsolat, mialatt felépítik, használják, és újra lebontják. Ez az állapotsorozat érvényes a modemes és a router-router közötti kommunikációra is.

Amikor a vonal *HALOTT*, nincs fizikai rétegbeli vivő, nincs fizikai rétegbeli kapcsolat. Miután a fizikai kapcsolat felépült, a vonal a *FELÉPÜLT* állapotba kerül. Ezen a ponton kezdődik meg az opciók egyeztetése (LCP-vel), ami – ha sikeres – a *HITELESÍTÉS*-be vezet. Itt a két fél ellenőrizheti egymás kilétét, ha van rá igény. Amikor a vonal a *HÁLÓZAT* fázisba lép, a megfelelő NCP protokoll meghívásával megtörténik a hálózati réteg felkonfigurálása. Ha ez sikeres, elérünk a *NYITVA* állapotba, amiben lebonyolódhat az adatátvitel. Amikor ez befejeződik, a vonal a *LEZÁRT* fázisba megy, és amikor a vivőt elengedi, vissza a *HALOTT* fázisba.

Az LCP arra szolgál, hogy a felek megegyezzenek az adatkapcsolati protokollopciókban a *FELÉPÜLT* fázis során. Az LCP protokoll nem magukkal az opciókkal foglalkozik, hanem a megegyezés mechanizmusával. Biztosítja, hogy a kezdeményező folyamat tehessen egy javaslatot, és a válaszoló folyamat azt elfogadhassa vagy elutasíthassa egészben vagy részben. Szintén biztosítja, hogy a két folyamat kipróbálhassa a vonal minőségét, hogy lássa, hogy elég jónak találja-e a kapcsolat felépítéséhez. Végül, az LCP protokoll azt is lehetővé teszi, hogy a kapcsolatot le lehessen bontani, amikor már nincs rá szükség.

Tizenegyféle LCP csomag típust definiál az RFC 1661. Ezeket a 3.29. ábrán soroltuk fel. A négy *Konfigurál*-típus lehetővé teszi a kezdeményezőnek (K), hogy javasoljon opcióértékeket és a válaszolónak (V), hogy elfogadja, vagy elutasítsa azokat. Ez



3.28. ábra. *Egy kapcsolat felépítésének és lebontásának egyszerűsített fázisdiagramja*

Név	Irány	Leírás
Konfigurál-kérés	K → R	Javasolt opciók és értékek listája
Konfigurál-nyugta	K ← R	Minden opció elfogadva
Konfigurál-negatív nyugta	K ← R	Néhány opció nincs elfogadva
Konfigurál-elutasítás	K ← R	Néhány opcióról nem lehet egyezkedni
Lezár-kérés	K → R	A kapcsolat lebontásának kérése
Lezár-nyugta	K ← R	Rendben, a kapcsolat lebontva
Kód-elutasítás	K ← R	Ismeretlen kérés érkezett
Protokoll-elutasítás	K ← R	Ismeretlen protokollt kért
Visszhang-kérés	K → R	Küldd vissza ezt a keretet
Visszhang-válasz	K ← R	Itt a visszaküldött keret
Eldobás-kérés	K → R	Dobd el ezt a keretet (teszteléshez)

3.29. ábra. *Az LCP csomag típusok*

utóbbi esetben a válaszoló tehet alternatív javaslatot vagy bejelentheti, hogy egyáltalán nem akar egyezkedni bizonyos opciókról. Az egyeztetendő opciók és értékeik az LCP csomag részei.

A *Lezár*-kódok arra használatosak, hogy a kapcsolatot lebontsuk, amikor már nincs rá szükség. A *Kód-elutasítás* és a *Protokoll-elutasítás* kódokat arra használja a vevő, hogy jelezze, hogy olyan valamit kapott, amit nem ért. Ez jelentheti azt, hogy észrevétlen átviteli hiba történt, de sokkal valószínűbb, hogy a kezdeményező és a válaszoló az LCP protokoll különböző verzióit futtatja. A *Visszhang*-típusok a vonal tesztelésére használatosak. Végül az *Eldobás-kérést* hibakeresésre használjuk. Ha bármelyik végpontnak gondjai vannak a bitek vonalra juttatásával, a programozó ezt a típust használhatja a teszteléshez. Ha ennek sikerül átjutnia a vonalon, a vevő csak eldobja, ahelyett, hogy valami olyan dolgot csinálna, ami a tesztelőt megzavarná.

Azok az opciók, amelyekben a két fél megegyezhet, magukban foglalják az adat mező maximális méretének beállítását, a hitelesítés engedélyezését, az alkalmazandó protokoll kiválasztását, a normális működés közbeni vonal minőségi monitorozását és különféle fejléctömörítési opciók kiválasztását.

Általánosságban keveset lehet mondani az NCP protokollokról. Mindegyik valamelyik hálózatiréteg-protokollhoz kapcsolódik, és lehetővé teszi az adott protokollra vonatkozó konfigurációs kérések megbeszélését. Az IP számára például a dinamikus címhozzárendelés a legfontosabb lehetőség.

3.6.3. Az ATM adatkapcsolati rétege

Most itt az ideje, hogy elkezdjük utazásunkat az 1.30. ábrán látható ATM protokoll rétegek között. Az ATM fizikai rétege nagyjából lefedi az OSI fizikai és adatkapcsolati rétegét. A fizikai közegfüggő (physical medium dependent – PMD) alréteg az OSI fizikai rétegéhez hasonlóan működik, az átviteli konvergencia (transmission convergency – TC) alréteg pedig adatkapcsolati funkcionalitással rendelkezik. Nincsen az ATM-nek speciális fizikai rétege. Ehelyett az ATM cellákat SONET, FDDI és más átviteli rendszerek szállítják. Emiatt itt a TC alréteg adatkapcsolati funkcióira koncentrálunk, de később megtárgyaljuk az alsóbb alréteg felé biztosított interfészt is néhány szempontból.

Amikor egy felhasználói program előállít egy elküldendő üzenetet, az miközben végighalad lefelé az ATM protokollkészleten, fej- és farokrészeket kap és átesik a cellákra bontáson. Végül is a cellák elérik a TC alréteget, ahol továbbításra kerülnek. Nézzük meg, mi történik velük útjuk során.

Cellatovábbítás

Az első lépés a fejrész ellenőrző összegének előállítása. Mindegyik cella tartalmaz egy 5 bájtos fejrészt, mely 4 bájt virtuális áramkör és vezérlő információból és az azt követő 1 bájtos ellenőrző összegből áll. Bár a fejrész tartalma nem érdekes a TC alréteg szempontjából, a kíváncsi olvasók vethetnek lopva egy pillantást az 5.62. ábrára. Az ellenőrző összeg csak az első négy fejrész bájtot fedi le, az adat mezőt nem. Az ellenőrző összeg a 32 fejrész bit $x^8 + x^2 + x + 1$ polinommal való osztás után kapott maradékából áll. Ehhez a 01010101 konstanst adják hozzá a többségében 0 bitet tartalmazó fejrészekkel szembeni robusztusság biztosítására.

A döntést, miszerint csak a fejrészt védik ellenőrző összeggel azért hozták, hogy csökkentsék a fejrészhiba miatt történő hibáscsomag-kézbesítések valószínűségét, de mentesüljenek a sokkal nagyobb adat mező ellenőrző összeggel történő ellátásának költségétől. Ennek elvégzése a felsőbb rétegekre van bízva, ha igényük van rá. Sok valós idejű alkalmazásnál, mint például a hang- és a videoátvitelnél időnként néhány bit elvesztése elfogadható (bár néhány tömörítési eljárásnál minden keret egyenlő, de néhány keret még egyenlőbb). Mivel csak a fejrészt fedi le, a 8 bites ellenőrző összeg mezőt **HEC-nek (Header Error Control – fejrész hibavédelem)** nevezik.

A főszerepet játszó tényező ebben az ellenőrzőösszeg-képzési elgondolásban az a tény volt, hogy az ATM-et fényvezető üvegszál feletti használatra tervezték és a fényvezető üvegszál nagyon megbízható. Ezen kívül egy, az Egyesült Államok telefonhálózatán végzett átfogó vizsgálat kimutatta, hogy normál működés során az üvegszálon fellépő összes hiba 99,64%-a egybites (AT&T és Bellcore, 1989). A HEC módszer kijavítja az összes egybites hibát és sok több-bites hibát is jelez. Ha feltételezzük, hogy az egybites hibák valószínűsége 10^{-8}, akkor a jelezhető több-bites hibával rendelkező fejrészt tartalmazó keret előfordulásának valószínűsége körülbelül 10^{-13}. Annak a valószínűsége, hogy egy cella átcsúszik federítetlen fejrészhibával, körülbelül 10^{-20}, ami azt jelenti, hogy OC–3-as sebességnél minden 90 000 évben jut át egy rossz cellafejléc. Bár ez hosszú időnek tűnhet, ha egyszer a Földön lesz, mondjuk 1 billió ATM telefon, melyeket az idő 10%-ában használnak, több mint 1000 rossz cellafejrész marad észrevétlen évente.

Azoknak az alkalmazásoknak a számára, amelyek megbízható átvitelt igényelnek az adatkapcsolati rétegben, Shacham és McKenney (1990) kifejlesztett egy eljárást, amelyben az egymást követő cellák sorozatát KIZÁRÓ VAGY kapcsolatba hozzuk.

Az eredményt, egy egész cellát, hozzáfűzzük a sorozathoz. Ha egy cella elveszik vagy nagyon megsérül, visszaállítható a rendelkezésre álló információból.

Ha a HEC előállítása megtörtént, és beszúrtuk a cellafejrészbe, a cella kész a továbbításra. Az átviteli közegek két kategóriába sorolhatók: aszinkron és szinkron. Amikor aszinkron közeget alkalmazunk, a kész cella azonnal elküldhető, nincsenek időzítési megkötések.

Szinkron médiumnál a cellákat az előzetesen definiált időzítési sémának megfelelően kell továbbítani. Ha nem áll rendelkezésre adatcella, amikor szükséges, akkor a TC alrétegnek ki kell találnia egyet. Az ilyen cellákat **üresjárati celláknak (idle cells)** nevezzük.

A nem adatcellák egy másik fajtája az **OAM (Operation And Maintenance – vezérlő és karbantartó)** cella. Az OAM cellákat az ATM kapcsolók használják a rendszer működtetéséhez szükséges vezérlő és egyéb információk kicserélésére. Az OAM celláknak van még másik szerepük is. Például a 155,52 Mb/s-os OC–3 sebesség megegyezik a SONET bruttó adatsebességével, de egy STM–1 keretben van összesen 10 oszlopnyi overhead a 270-ből, így a SONET-ben az adat mező csak $260/270 \times 155{,}52$ Mb/s, azaz 149,76 Mb/s sebességű. Hogy elkerülje a SONET elárasztását, egy SONET-et használó ATM forrás minden 27. elküldött cellájának OAM cellának kell lennie, az adatsebesség 155,52 Mb/s 26/27-ed részére való lassítása érdekében, ami pontosan megegyezik a SONET sebességével. Az ATM kimeneti sebességnek az ATM alatt levő átviteli rendszerhez való igazítása a TC alréteg fontos feladata.

A vevő oldalán az üresjárati cellákat a TC alréteg dolgozza fel, de az OAM cellákat átadja az ATM rétegnek. Az OAM cellákat az különbözteti meg az adatcelláktól, hogy a fejrész első három bájtja nulla, ami az adatcelláknál nem megengedett. A negyedik bájt az OAM cella típusát határozza meg.

A TC alréteg egy másik fontos feladata az alatta levő átviteli rendszer számára a keretezési információ előállítása, ha erre szükség van. Például lehetséges, hogy egy ATM videokamera előállítja az ATM cellák sorozatát, és csak elküldi a vezetéken, de lehet, hogy a SONET kereteket is megcsinálja az adat mezőbe ágyazott ATM cellákkal. Az utóbbi esetben a TC alrétegnek elő kell állítania a SONET keretezést, és bele kell csomagolnia az ATM cellákat, ami nem teljesen triviális, hiszen a SONET adat mező mérete nem egész számú többszöröse az 53 bájtos cellaméretnek.

Bár a telefontársaságok egyértelműen SONET-et szándékoznak használni az ATM átviteli rendszereként, az ATM más rendszerek adat mezőire való leképezését szintén definiálták, és újabbak is kidolgozás alatt állnak. Létezik például leképezés a T1-re, T3-ra és az FDDI-ra is.

Cellavétel

Az adó oldalon a TC alréteg feladata az, hogy vegye a cellák sorozatát, hozzátegye a HEC-t mindegyik cellához, az eredményt bitsorozattá alakítsa és hozzáigazítsa a bitfolyam sebességét a fizikai átviteli rendszeréhez töltelék OAM cellák beszúrásával. A vevő oldalon a TC alréteg pontosan a fordítottját csinálja. Veszi a bejövő bitfolyamot, megkeresi a cellahatárokat, ellenőrzi a fejrészeket (eldobja az érvénytelen fejrésszel

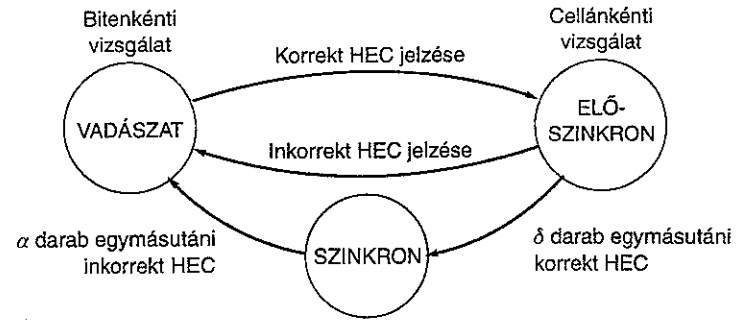
rendelkező cellákat), feldolgozza az OAM cellákat és az adatcellákat továbbítja az ATM rétegnek.

A legnehezebb dolog a cellahatárok megtalálása a bejövő bitfolyamban. Bit szinten a cella csak egy $53 \times 8 = 424$ hosszú bitsorozat. Nincs 01111110 jelző bájt a cella elején és végén, mint a HDLC-ben. Valójában egyáltalán nincsenek jelzők. Hogyan lehet ilyen körülmények között felismerni a cellahatárokat?

Néhány esetben a fizikai réteg segítséget nyújt. A SONET-nél például a cellákat hozzá lehet igazítani a szinkron adat mező borítékhoz, így a SONET fejrészben levő SPE mutató az első teli cellára mutathat. Néha azonban a fizikai réteg nem nyújt semmilyen támogatást a keretezéshez. Mi van ilyenkor?

A trükk a HEC felhasználása. A bejövő bitek egy a TC alrétegben működő léptetőregiszterbe kerülnek, mely 40 bites és a bitek a bal szélén lépnek be és a jobb szélén lépnek ki. A TC alréteg megvizsgálja a 40 bitet, hogy potenciálisan érvényes fejrész-e. Ha az, a jobb szélső 8 bit egy érvényes HEC a bal szélső 32 bitre nézve. Ha ez nem áll fenn, a puffer nem tartalmaz érvényes fejrészt, ekkor a pufferben levő biteket eggyel jobbra csúsztatjuk, ami egy bit kiesését és a bal szélen egy új bit belépését okozza. Ezt az eljárást addig ismételjük, amíg egy érvényes HEC-t nem találunk. Ekkor a cellahatárokat már ismerjük, hiszen a léptetőregiszter egy érvényes fejrészt tartalmaz.

Ezzel a heurisztikával a probléma az, hogy a HEC csak nyolc bit széles. Tetszőleges adott léptetőregiszternél, még a véletlen biteket tartalmazónál is, annak a valószínűsége, hogy érvényes HEC-t találunk 1/256, ami meglehetősen nagy érték. Önmagában alkalmazva ez az eljárás túl gyakran észleli helytelenül a cellafejrészeket.



3.30. ábra. *A cellahatárt kereső heurisztika*

Hogy megnöveljék a felismerő algoritmus pontosságát, a 3.30. ábrán látható véges állapotú automatát alkalmazzák. Három állapot van: *VADÁSZAT*, *ELŐSZINKRON* és *SZINKRON*. A *VADÁSZAT* állapotban a TC alréteg egyenként lépteti a biteket a léptetőregiszterbe érvényes HEC után kutatva. Amint talált egyet, a véges állapotú automata átmegy az *ELŐSZINKRON* állapotba, ami azt jelenti, hogy valószínűleg megtalálta a cellahatárt, de ezt még ellenőrizni kell. Ekkor beléptet 424 bitet (53 bájtot) anélkül, hogy megvizsgálná őket. Ha jól találta el a cellahatárt, akkor a léptetőregiszternek most egy másik érvényes cellát kell tartalmaznia, így most még egyszer lefuttatja a

HEC algoritmust. Ha a HEC helytelen, a TC visszalép a *VADÁSZAT* állapotba és bitről bitre, tovább keresi a fejrészt, amelynek helyes a HEC-je.

Ha a második HEC szintén helyes, a TC lehet, hogy jó úton jár, ezért beléptet még 424 bitet és újra próbálkozik. A fejrészek ilyen vizsgálata egészen addig folytatódik, amíg δ darab helyes fejrészt nem talál egymás után, amikor is azt feltételezi, hogy szinkronba került a callafolyammal, és a *SZINKRON* állapotba lép, hogy megkezdhesse a normális működést. Vegyük észre, hogy annak a valószínűsége, hogy véletlenül a *SZINKRON* állapotba kerül a rendszer teljesen véletlenszerű bitfolyam esetén $2^{-8\delta}$, ami tetszőlegesen kicsivé tehető elég nagy δ-t választva. A nagy δ-ért azonban a hosszabb szinkronizációs idővel kell fizetni.

Ezen kívül a szinkron elvesztése (vagy indulás) utáni újraszinkronizációhoz, a TC alrétegnek szüksége van egy heurisztikára ahhoz, hogy meghatározza, hogy mikor vesztette el ténylegesen a szinkront (például egy bit bitfolyamba való beszúródásakor, vagy abból való törlődésekor). Értelmetlen lenne rögtön feladni, ha egy HEC helytelen, hiszen a legtöbb hiba olyan, hogy egy bit értéke megváltozik, s nem új bitek adódnak az értékekhez, vagy vesznek el. A legokosabb eljárás ebben az esetben az, hogy eldobjuk a rossz fejléccel rendelkező cellát, és reméljük, hogy a következő jó lesz. Ha azonban α darab HEC egymás után rossz, a TC alrétegnek arra a követetésre kell jutnia, hogy kiesett a szinkronból, és vissza kell térnie a *VADÁSZAT* állapotba.

Bár elég valószínűtlen, mégis elképzelhető, hogy egy rosszindulatú felhasználó megpróbálja becsapni a TC alréteget azzal, hogy a HEC algoritmust imitáló adatmintáksorozatát teszi az adat teher mezőbe. Ekkor, ha a szinkront elvesztenénk, lehet, hogy rossz helyre állítanánk vissza. Hogy ezt a trükköt sokkal nehezebbé tegyék, az adat mező bitjeit összekeverik a továbbítás előtt és visszakeverik a vétel után.

Mielőtt elhagynánk a TC alréteget, egy megjegyzésnek helyet kell adnunk. Az a mechanizmus, amelyet a cellák ábrázolására választottak megköveteli, hogy a TC alréteg értse és használja a felette levő ATM réteg fejrészét. Az, hogy az egyik rétegnek használnia kell egy felsőbb réteg fejrészét, a protokolltervezés alapszabályainak totális áthágása. A réteges protokollok alapötlete pont az, hogy mindegyik réteget függetlenné tesszük a felette levőktől. Például lehetségesnek kellene lennie, hogy megváltoztassuk az ATM réteg fejrészformátumát anélkül, hogy az a TC alrétegre bármilyen hatással legyen. Az ilyen változtatás azonban lehetetlen a cellák ábrázolásának ilyenfajta megoldása mellett.

3.7. Összefoglalás

Az adatkapcsolati réteg feladata az, hogy a fizikai réteg által kínált nyers bitfolyamot átalakítsa keretfolyammá, amit a hálózati réteg használ. Változatos keretezési eljárások használatosak, ideértve a karakterszámlálást, karakterbeszúrást és a bitbeszúrást. Az adatkapcsolati protokollok biztosíthatnak hibavédelmet a sérült vagy elveszett keretek újraküldéséhez. Ahhoz, hogy megakadályozzuk, hogy a gyors adó elárassza a lassú vevőt, az adatkapcsolati protokoll biztosíthat forgalomszabályozást is. A csúszóablakos mechanizmust széles körben alkalmazzák a hibavédelem és a forgalomszabályozás kényelmes, egyesített megoldására.

A csúszóablakos protokollokat a küldő és a vevő ablakának mérete szerint kategorizálhatjuk. Ha mindkettő 1, a megáll-és-vár protokollal állunk szemben. Ha a küldő ablaka nagyobb, mint 1, például azért hogy megakadályozza az adó blokkolódását egy nagy késleltetési idejű vonal esetén, a vevőt lehet úgy programozni, hogy eldobja a sorrendben következőn kívüli összes keretet (5. protokoll), vagy úgy hogy pufferelje a sorrenden kívüli kereteket addig, amíg szükség nem lesz rájuk (6. protokoll).

A protokollok változatos technikák alkalmazásával modellezhetők, ami segít megmutatni a helyességüket (vagy helytelenségüket). Általában a véges állapotú automata és a Petri-gráf modelleket használják erre a célra.

Sok hálózat a bitalapú protokollok – SDLC, HDLC, ADCCP vagy LAPB – közül valamelyik protokollt alkalmazza az adatkapcsolati rétegben. Ezek a protokollok mind jelző bájtokat használnak a keretek határolására, és bitbeszúrást annak megakadályozására, hogy a jelző bájtok előforduljanak az adatok között. Mindegyik csúszóablakot használ a forgalomszabályozáshoz. Az Interneten az SLIP-t és a PPP-t használják adatkapcsolati protokollként. Az ATM rendszernek megvan a saját egyszerű protokollja, amely csak a legminimálisabb hibajavítást végzi, forgalomszabályozást egyáltalán nem végez.

Feladatok

1. Egy felsőbb rétegbeli üzenetet 10 keretre osztunk, és mindegyiknek 80% esélye van arra, hogy sértetlenül megérkezzen. Ha az adatkapcsolati réteg semmilyen hibavédelmet nem végez, átlagosan hányszor kell az üzenetet elküldeni ahhoz, hogy az egész eljusson a vevőhöz?

2. Egy adatfolyam közepén, amelyre az ismertetett karakterbeszúrási algoritmust alkalmazzuk, a következő adatrészlet tűnik fel: DLE, STX, A, DLE, B, DLE, ETX. Milyen lesz a kimenet a karakterbeszúrás után?

3. Ha a 0111101111101111110 bitsorozatot bitbeszúrásnak vetjük alá, hogyan néz ki a kimeneti füzér?

4. Amikor bitbeszúrást alkalmazunk, lehetséges-e hogy egyetlen bit elveszése, beszúródása vagy módosulása olyan hibát okoz, amit az ellenőrző összeggel nem tudunk jelezni? Ha nem, miért nem? Ha igen, hogyan? Van-e szerepe itt az ellenőrző összeg hosszának?

5. El tud-e képzelni olyan körülményeket, amelyek között egy nyílthurkú protokoll (pl. egy Hamming-kód) előnyösebb lehet azoknál a visszacsatolás típusú protokolloknál, amelyeket ebben a fejezetben ismertünk meg?

6. Egyetlen paritásbit által nyújtottnál nagyobb biztonságot akarunk elérni, így egy olyan hibaészlelő sémát alkalmazunk, amelyben két paritásbit van: az egyik a páros, a másik a páratlan bitek ellenőrzésére. Mekkora e kód Hamming-távolsága?

7. A hibák észlelésének egy lehetséges módja az, hogy az adatokat n sorból és k oszlopból álló blokkokban visszük át úgy, hogy minden egyes sorhoz és oszlophoz paritásbiteket adunk. Vajon ez az elrendezés tudja-e jelezni az összes egybites hibát? A kettősöket? A hármas hibákat?

8. Egy n soros, k oszlopos bitblokk vízszintes és függőleges paritásbiteket használ hibajelzésre. Tételezzük fel, hogy pontosan 4 bit állítódik át az átviteli hibák következtében. Vezessünk le egy képletet annak a valószínűségére, hogy a hiba észrevétlen marad.

9. Mekkora lesz a keletkező maradék, ha $x^7 + x^5 + 1$-et elosztjuk az $x^3 + 1$ generátorpolinommal?

10. Az adatkapcsolati protokollok majdnem mindig a farokrészbe teszik a CRC-t a fejrész helyett. Miért?

11. Egy csatornának 4 kb/s-os sebessége és 20 ms-os terjedési késleltetése van. Milyen keretméret-tartományra ad a megáll-és-vár stratégia legalább 50%-os hatékonyságot?

12. Egy 3000 km hosszú T1-es trönköt az ötödik protokoll segítségével 64 bájtos keretek átvitelére használunk. Ha a terjedési sebesség 6 μs/km, hány bitesnek kell lennie a sorszámnak?

13. Képzeljünk el egy olyan forgóablakos protokollt, amely annyi sorszámbitet használ, hogy körbefordulás soha nem fordul elő. Milyen összefüggésnek kell fennállnia a négy ablakszél és az ablakméret között?

14. Ha az ötödik protokollban a *between* eljárás az $a \leq b < c$ feltétel helyett az $a \leq b \leq c$ feltételt ellenőrizné, lenne-e ennek hatása a protokoll helyességére és hatékonyságára? Magyarázzuk meg!

15. A 6. protokollban, amikor egy keret megérkezik, ellenőrizzük, hogy a sorszám különbözik-e a várttól, és a *no_nak* igaz-e. Ha mindkét feltétel fennáll, egy NAK-ot küldünk. Ellenkező esetben elindítjuk a segéd időzítőt. Tételezzük fel, hogy az else ágat elhagyjuk. Befolyásolná-e ez a változtatás a protokoll helyességét?

16. Tételezzük fel, hogy a háromutasításos while ciklust a 6. protokoll vége környékéről elhagyjuk. Befolyásolja-e ez a protokoll helyességét, vagy csak a teljesítőképességét? Magyarázza meg válaszát!

17. Tételezzük fel, hogy az ellenőrzőösszeg-hiba esetet elhagyjuk a 6. protokoll switch utasításából. Hogyan befolyásolná ez a változtatás a protokoll működését?

18. A hatodik protokollban a *frame_arrival*-t kezelő kódnak van egy NAK-okat kezelő része. Erre a részre akkor kerül a vezérlés, ha a bejövő keret NAK, és még egy másik feltétel is teljesül. Adjunk meg egy eseménysort, ahol ennek a másik feltételnek a jelenléte alapvető fontosságú!

19. Képzeljük el, hogy egy olyan vonal adatkapcsolati szoftverét írjuk meg, amelyben csak nekünk küldhetnek adatokat, de mi nem küldhetünk azon! A másik oldal HDLC-t használ 3 bites sorszámmal és hét keretes ablakkal. A hatékonyság érdekében annyi sorrenden kívüli keretet szeretnénk pufferelni, amennyit csak lehet, de a küldő oldalon nem módosíthatjuk a szoftvert. Lehetséges-e 1-nél nagyobb vevőablak alkalmazása úgy, hogy még garantálható legyen a protokoll hibamentes működése? Ha igen, akkor mekkora a legnagyobb, még biztonságosan használható ablakméret?

20. Tekintsük a 6. protokoll működését egy 1 Mb/s-os hibamentes vonalon. A maximális keretméret 1000 bit. Az új csomagok egy másodperces időközönként generálódnak. Az időzítési intervallum 10 ms. Ha elhagynánk a speciális nyugtázási időzítéseket, akkor szükségtelen időtúllépések következnének be. Hányszor kellene elküldeni egy átlagos üzenetet?

21. A hatodik protokollban $MAX\_SEQ = 2^n - 1$. Míg ez a feltétel nyilvánvalóan kívánatos a fejrészbitek hatékony használata miatt, azt nem mutattuk meg, hogy egyben alapvető fontosságú is. Vajon a protokoll helyesen működne-e $MAX\_SEQ = 4$ esetén?

22. 1000 bites kereteket küldünk egy 1 Mb/s-os műholdas csatornán. A nyugtákat mindig ráültetjük az adatkeretekre. A fejrészek nagyon rövidek. Hárombites sorszámokat használunk. Mennyi a maximálisan elérhető csatornakihasználtság a

    (a) megáll-és-vár,

    (b) 5. protokoll,

    (c) 6. protokoll

    esetében?

23. Számítsuk ki, hogy a sávszélesség mekkora hányada megy veszendőbe az overhead (fejrészek és újraadások) miatt a hatodik protokoll használata esetén, egy erősen terhelt 50 kb/s-os műholdas csatornán, melyen az adatkeretek 40 fejrész- és 3960 adatbitet tartalmaznak! ACK keretek soha nem fordulnak elő. A NAK keretek 40 bitesek. Az adatkeretek hibaaránya 1%-os, a NAK kereteké elhanyagolható. A sorszámok 8 bitesek.

24. Vegyünk egy 64 kb/s-os műholdas csatornát, melyet 512 bájtos adatkeretek küldésére használunk az egyik irányban, és nagyon rövid nyugták jönnek vissza a másik irányban. Mennyi a maximális átviteli sebesség az 1, 7, 15 és a 127 ablakméretekhez?

25. Egy 100 km hosszú kábel T1-es adatsebességgel működik. A terjedési sebesség a kábelen a fénysebesség 2/3-a. Hány bit fér a kábelre?

26. Rajzoljuk újra a 3.21. ábrát egy duplex csatornára, amely soha nem veszt keretet! Lehetséges még protokollhiba?

27. Adjuk meg a tüzelési sorrendjét a 3.23. ábra Petri-gráfjának, amely a 3.20. ábra (000), (01A), (01–), (010), (01A) állapotsorozatának felel meg! Írjuk körül szavakkal, hogy mit reprezentál a sorozat!

28. Ha adottak az $AC \rightarrow B$, $B \rightarrow D$, $CD \rightarrow E$ és $E \rightarrow CD$ átmenetre vonatkozó szabályok, rajzoljuk fel az így leírt Petri-gráfot! A Petri-gráfból rajzoljuk fel az ACD kezdeti állapotból elérhető véges állapotú automatát! Melyik jól ismert elméleti számítástechnikai fogalom alkalmazza ezt az átmenetre vonatkozó szabálymodellt?

29. A PPP a HDLC-n alapul, amely bitbeszúrást alkalmaz, hogy megakadályozza a hasznos teher mezőben véletlenül előforduló jelző bájtok által okozott kavarodást. Adjunk legalább egy okot, ami miatt a PPP mégis inkább karakterbeszúrást alkalmaz!

30. Mennyi a minimális overhead egy IP csomag PPP-vel való elküldésekor? Csak a PPP által okozott rezsit vegyük figyelembe, az IP fejrészét ne!

31. Vegyük az ATM cellaábrázoló heurisztikát $\alpha = 5$, $\delta = 6$ és 10^{-5} bitenkénti hiba mellett! Ha a rendszer egyszer szinkronizálódott, milyen hosszú ideig marad így az esetleges bithibák ellenére? Tételezzük fel, hogy a vonal OC–3-as sebességgel működik.

32. Írjunk egy programot, amely sztochasztikusan szimulálja egy Petri-gráf viselkedését! A program olvassa be az átmenetre vonatkozó szabályokat, valamint egy olyan állapotokból álló listát, amelyek a hálózati réteg új csomag kibocsátását vagy új csomag elfogadását azonosítják! A programnak a kezdeti állapotból kiindulva, amelyet szintén be kell olvasnia, véletlenszerűen ki kell választania a megengedett átmeneteket, és ezeket végre kell hajtania, így kell ellenőriznie azt, hogy az egyik hoszt elfogad-e két üzenetet anélkül, hogy a másik közben kibocsátott volna egy újat!

4. A közegelérési alréteg

Ahogy már az első fejezetben rámutattunk, a hálózatok két kategóriába sorolhatóak: vannak, amelyek kétpontos összeköttetést, és vannak, amelyek adatszóró csatornát használnak. Ez a fejezet az adatszóró hálózatokkal, és az ilyen hálózatokon használható protokollokkal foglalkozik.

Minden adatszóró hálózat esetén a kulcskérdés az, hogy versenyhelyzetben hogyan állapítható meg melyik állomás nyerje el a jogot a csatorna használatára. Ahhoz, hogy a problémát tisztábban lehessen látni, vegyünk egy példát: képzeljünk el egy telefonos konferenciabeszélgetést, amelyben hat ember vesz részt, akik külön telefonkészülékeket használnak. A konferencia során minden készülék között van kapcsolat, így mindenki hallja a másikat, és bárkihez tud szólni. Ilyen szituációkban sűrűn fordul elő az, hogy amikor valaki befejezi a mondandóját, egyszerre ketten vagy többen is megszólalnak, ami teljes kavarodáshoz vezet. Ez a zűrzavar nem jelentkezne egy szemtől szembe típusú találkozón, hiszen ekkor külső jelekkel, például a megbeszélés során kézfeltartással jelezhetnék a résztvevők felszólalási szándékukat. Viszont akkor, ha csak egyetlen csatorna áll rendelkezésre, sokkal nehezebb annak eldöntése, hogy ki használhatja a csatornát következőként. A feladat megoldására rengeteg protokoll ismert, és a fejezet ezekről próbál áttekintést nyújtani. Az irodalomban az adatszóró csatornákra (broadcast channel) sokszor hivatkoznak **többszörös elérésű csatorna (multiaccess channel)**, illetve **véletlen elérésű/hozzáférésű csatorna (random access channel)** megnevezéssel is.

Az adatkapcsolati réteg egy alrétegéhez, a **MAC** alréteghez **(Medium Access Control)** tartoznak azok a protokollok, amelyek a közeg használatának vezérléséért felelősek. A MAC alréteg különösen fontos szerepet tölt be a LAN hálózatokban, amelyek közül szinte mindegyik többszörös elérésű csatornára építi kommunikációját. Ezzel szemben a műholdas rendszerek kivételével a WAN hálózatok kétpontos összeköttetésekből állnak össze. Mivel a többszörös hozzáférésű csatornák és a LAN hálózatok ilyen szoros kapcsolatban állnak, ebben a fejezetben általánosan esik majd szó a LAN hálózatokról, valamint szerepelni fognak műholdas rendszerek és néhány egyéb adatszóró hálózat is.

Technikailag a MAC alréteg az adatkapcsolati réteg alsó részét képezi, így logikai-

lag a 3. fejezetben részletezett kétpontos protokollok előtt kellett volna ismertetni. Mindazonáltal a legtöbb ember számára egyszerűbb a több résztvevő számára készült protokollokat megérteni azután, ha már megismerte a két résztvevő számára készült protokollokat, ezért kis mértékben eltértünk a tényleges alulról felfelé történő bemutatástól.

4.1. A csatornakiosztás problémája

Ez a fejezet elsősorban azzal foglalkozik, hogyan lehet egy adatszóró csatornát a versenyző felhasználók között kiosztani. Először általánosan fogjuk vizsgálni a statikus és dinamikus módszereket, majd megvizsgálunk néhány konkrét algoritmust.

4.1.1. Statikus csatornakiosztás LAN-ok és MAN-ok esetén

Hagyományosan, például telefontrönkök esetében, frekvenciaosztásos nyalábolással (Frequency Division Multiplexing, FDM) oldják meg a csatornakiosztást a versengő felhasználók között. Ha N felhasználó van, a sávszélességet N darab egyenlő méretű sávra osztják (lásd 2.24. ábra), majd minden felhasználót hozzárendelnek az egyik frekvenciasávhoz. Miután minden felhasználónak saját frekvenciasávja van, nem zavarhatják egymást. Az FDM egyszerű és hatékony kiosztási mechanizmus olyan esetekben, amikor kevés, fix számú felhasználó van, azok viszont nagy (pufferelt) forgalmi igénnyel rendelkeznek (pl. egy szállítmányozó cég elosztó irodája).

Azokban az esetekben azonban, amikor a küldő felek száma nagy és folyamatosan változik, vagy az adatforgalom löketes jellegű, felmerül néhány probléma az FDM alkalmazásával. Ha a frekvenciaspektrumot N részre osztottuk, viszont egyszerre éppen N-nél kevesebb felhasználó szeretne kommunikálni, az értékes spektrum jókora hányada veszik kárba. Ha viszont N-nél több felhasználó szeretne kommunikálni, szabad sávok hiányában néhányukat a rendszernek vissza kell utasítania még akkor is, ha a frekvenciasávval rendelkező felhasználók közül néhányan nem is küldenek vagy fogadnak adatokat.

A meglevő csatorna statikus alcsatornákra bontása természetesen még akkor sem lehet hatékony megoldás, ha feltételezzük, hogy a felhasználók száma állandóan N marad. Az alapvető probléma az, hogy amíg a felhasználók nem forgalmaznak, a számukra kijelölt frekvenciatartomány egyszerűen elveszik, mivel ők nem használják, mások pedig nem férhetnek hozzá. Ráadásul a legtöbb számítógépes rendszerben az adatforgalom alapvetően löketes jellegű (a maximális és átlagos forgalom 1000:1 aránya elég általános), így a csatornák többsége az idő túlnyomó részében kihasználatlan marad.

A statikus FDM alacsony hatékonysága könnyen belátható egy, a sorbanállási elméleten alapuló egyszerű számítással. Számítsuk ki az igények által a rendszerben eltöltött időt (T). A csatorna kapacitása legyen C b/s, átlagosan λ keret/másodperc intenzitással érkezzenek elküldendő adatok, és a keretek hosszát $1/\mu$ bit/keret várható értékű exponenciális eloszlás határozza meg.

$$T = \frac{1}{\mu C - \lambda}$$

Most vágjuk a csatornát N darab független alcsatornára, amelyek közül mindegyik C/N b/s kapacitással rendelkezik. Ekkor az átlagos érkezési intenzitás mindegyik alcsatornán λ/N lesz. Újraszámolva a T-t a következő eredményt kapjuk.

$$T_{\text{FDM}} = \frac{1}{\mu(C/N) - (\lambda/N)} = \frac{N}{\mu C - \lambda} = NT \tag{4.1}$$

Az igények által a rendszerben eltöltött átlagos idő FDM alkalmazása esetén N-szer nagyobb annál, mintha az összes keret valamilyen varázslatos módon egy nagy központi sorba rendeződött volna.

Az FDM-re megállapított argumentumok pontosan megegyeznek az időosztásos nyalábolás (Time Division Multiplexing, TDM) esetében is. Ekkor minden egyes felhasználót statikusan hozzárendelnek egy-egy időréshez. Ha egy felhasználó nem használja ki a számára fenntartott időrést, akkor az egyszerűen üres marad. Mivel semmilyen hagyományos statikus hozzárendelési módszer nem képes jó hatásfokkal dolgozni löketes forgalom esetén, most vizsgáljunk meg néhány dinamikus megoldást.

4.1.2. Dinamikus csatornakiosztás LAN-ok és MAN-ok esetén

Mielőtt a fejezet által tárgyalt rengeteg csatornakiosztási módszer közül az elsővel el kezdenénk foglalkozni, megéri kis időt szakítani a csatornakiosztás problémájának pontos megfogalmazására. A munkánk során a következő öt kulcsfontosságú feltételezéssel fogunk élni.

1. **Állomás modell.** A modell N független **állomást (station)** feltételez (számítógépet, telefont, személyi hívót stb.), amelyek közül mindegyik egy program vagy felhasználó közreműködésével továbbítandó kereteket generál. Annak a valószínűsége, hogy egy Δt időtartam alatt keret generálódik $\lambda \Delta t$, ahol λ egy konstans (az új keretek érkezési intenzitása). Ha egy állomás generált egy keretet, akkor blokkolt állapotban marad mindaddig, amíg a keretet sikeresen nem továbbította.

2. **Egyetlen csatorna feltételezése.** Mindenféle kommunikációhoz egyetlen csatorna vehető igénybe. Minden állomás képes ezen adatokat továbbítani, illetve adatokat fogadni. A hardver megvalósítás az állomásokat egyenrangúnak tekinti, bár a protokoll szoftver prioritást rendelhet hozzájuk.

3. **Ütközés feltételezése.** Ha két keretet egy időben továbbítanak, akkor azok átlapolódnak, és az eredményül kapott jel értelmezhetetlenné válik. Az ilyen eseményt **ütközésnek (collision)** nevezzük. Az ütközéseket minden állomás képes észlelni. Az ütközésbe került kereteket később újra kell küldeni. Az ütközéseken kívül semmilyen egyéb hiba nem léphet fel.

4a. **Folytonos idő.** A keretek továbbítása bármelyik időpillanatban megkezdődhet. Nincs központi óra, amely az időt diszkrét intervallumokra osztaná.

4b. **Diszkrét idő (slotted time).** Az idő diszkrét intervallumokra (időrés) van osztva. A keretek továbbítása mindig csak az időrések elején kezdődhet meg. Egy időrés 0, 1 vagy több keretet tartalmazhat, és ennek megfelelően nevezzük az időrést üresnek, sikeresnek vagy ütközésesnek.

5a. **Vivőjel-érzékelés (carrier sense).** Az állomások meg tudják állapítani, hogy a csatorna foglalt-e, mielőtt megpróbálnák használni. Ha a csatornát foglaltnak érzékelik, akkor egyetlen állomás sem próbálja meg használatba venni, amíg üressé nem válik.

5b. **Nincs vivőjel-érzékelés.** Az állomások nem tudják megvizsgálni a csatornát, mielőtt megpróbálnák használatba venni, így egyszerűen elkezdenek adni. Csak ezután tudják eldönteni, hogy az átvitel sikeres volt-e vagy sem.

Néhány megjegyzés a feltételezésekhez. Az első azt mondja, hogy az állomások függetlenek egymástól, és az igények állandó intenzitással generálódnak. Implicit módon azt is feltételezi, hogy egyetlen program vagy felhasználó vezérli a munkaállomásokat, így amíg az állomás blokkolt állapotban van, nem keletkezik újabb igény rajta. Bonyolultabb modellek megengednek több programot futtató állomásokat, amelyek így blokkolt állapotban is képesek igényeket generálni, de az ilyen állomások analízise sokkal komplexebb feladat.

A probléma gyökerét az egyetlen csatorna feltételezése képezi, ugyanis emiatt nincs semmilyen külső kommunikációs lehetőség. Az állomások nem képesek feltenni a kezüket, hogy a tanár felszólítsa őket.

Szintén alapvető az ütközések feltételezése még akkor is, ha néhány rendszerben (szórt spektrumú rendszerek) ez a feltételezés gyengébb, és így meglepő eredményeket képes produkálni. Néhány LAN hálózat esetében pedig, mint például a Token Ring rendszerek, olyan mechanizmust használnak a versenyhelyzetek megelőzésére, amely segítségével elkerülhetők az ütközések is.

Az időt illetően két különböző feltételezés létezik: lehet folytonos, illetve diszkrét. Bizonyos rendszerek az egyiket, mások pedig a másikat használják, ezért mindkettőt tárgyalni és vizsgálni fogjuk. Természetesen egy adott rendszerre ezek közül csak az egyik állhat fenn.

Hasonlóan, egy hálózat lehet vivőjel-érzékeléses vagy nem. Általában a LAN-ok vivőjel-érzékelős hálózatok, de a műholdas rendszerek nem (a nagy terjedési késleltetés miatt). Vivőjel-érzékelést támogató hálózatokon az állomások az adást azonnal be tudják fejezni, ha érzékelik, hogy az másik adással ütközik. Itt jegyezném meg, hogy a „vivőjel" (carrier) szó egy villamos jelre utal, amely a kábeleken halad, és semmi köze nincs a távközlési szolgáltatóhoz (pl. a telefontársasághoz).

4.2. Többszörös hozzáférésű protokollok

Többszörös hozzáférésű csatornák kiosztására több algoritmus is ismert. A következő fejezetben megismerkedünk néhány jellegzetes protokollal az érdekesebbek közül, és példákat mutatunk be a használatukra.

4.2.1. ALOHA

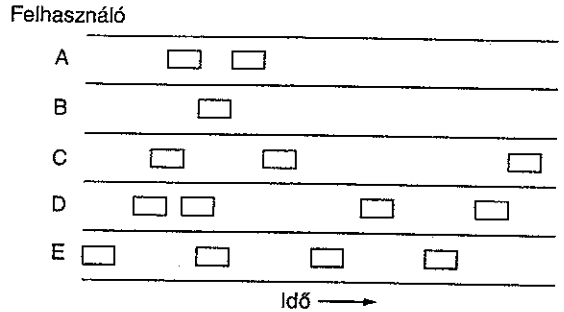
Az 1970-es években a Hawaii Egyetemen Norman Abramson munkatársaival tervezett egy új, elegáns módszert a csatornakiosztás problémájának megoldására. Munkájukat azóta sok kutató folytatta és fejlesztette tovább (Abramson, 1985). Bár Abramson ALOHA rendszernek nevezett munkáját földi telepítésű rádiós üzenetszóráshoz készítette, az alapötlete bármely olyan rendszerre alkalmazható, amelyben koordinálatlan felhasználók egyetlen megosztott csatorna hozzáférési jogáért versengenek.

Az ALOHA két változatát fogjuk vizsgálni: egyszerű és réselt. A különbség az, hogy az időt felosztjuk-e diszkrét időszeletekre, amelyekbe minden keretnek el kell férnie, vagy nem. Az egyszerű ALOHA nem igényel globális időszinkronizálást, míg a réselt ALOHA igen.

Egyszerű ALOHA

Az ALOHA rendszer alapötlete egyszerű: engedjük a felhasználót adni, amikor csak van továbbítandó adata. Természetesen ütközések lesznek, és ezek a keretek el fognak veszni. Az üzenetszórás visszacsatolásos jellege miatt azonban a küldő – a többi állomáshoz hasonlóan – figyelheti a csatornát, így meg tudja állapítani, hogy tönkrement-e a keret, vagy sem. Egy LAN esetén a visszacsatolás azonnali; egy műholdnál viszont a küldő csak 270 ms késleltetés után szerezhet tudomást az átvitel sikerességéről. Ha a keret megsérült, a küldő egyszerűen véletlenszerű ideig várakozik, majd ismét elküldi a keretet. A várakozási időnek véletlenszerűnek kell lennie, különben ugyanazok a keretek ütköznének újra és újra szabályos időközönként. Azokat a rendszereket, amelyekben a közös csatorna használata konfliktushelyzetek kialakulásához vezethet, **versenyhelyzetes (contention)** rendszereknek nevezzük.

Az ALOHA rendszerbeli keretgenerálás vázlatát a 4.1. ábra mutatja be. A kereteket egyforma hosszúnak vettük, mivel az ALOHA rendszerek átbocsátóképessége egyforma keretméretek esetén maximális.



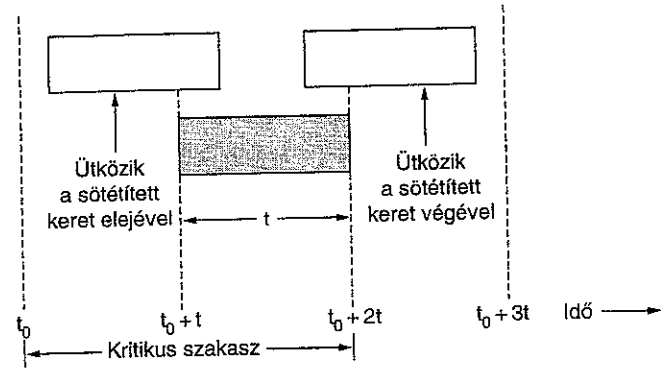
4.1. ábra. *Az egyszerű ALOHA rendszerben a keretek küldése tetszőleges időpillanatban kezdődhet meg*

Amikor ugyanabban az időpillanatban két keret is megpróbálja elfoglalni a csatornát, ütközés lép fel, és mindkét csomag megsérül. A két keret akkor is használhatatlanná válik, ha az egyik első bitje éppen hogy ütközik a másik utolsó bitjével, így később mindkét keretet újra kell majd küldeni. Az ellenőrző összeg nem képes megkülönböztetni (és nem is feladata) a teljes ütközést a részlegestől. Ami hibás, az hibás.

Nagyon érdekes kérdés a következő: milyen egy ALOHA csatorna hatékonysága. Tehát az elküldött keretek mekkora hányada képes épségben maradni ilyen kaotikus körülmények között? Először képzeljünk el egy végtelen számú interaktív felhasználói csoportot, ahol mindenki a terminálja (állomása) előtt ül. Egy felhasználó mindig két állapot közül az egyikben van: vagy gépel vagy várakozik. Kezdetben minden felhasználó gépel. Amikor egy sor elkészült, a felhasználó befejezi a gépelést és várja a visszajelzést. Ekkor az állomás egy keretbe helyezve továbbítja a sort, miközben figyeli a csatornát, hogy sikeresen átjutott-e a csomag. Ha sikeres volt az átvitel, a felhasználó visszajelzést kap erről, majd tovább gépel. Ha az átvitel nem sikerül, a felhasználó tovább várakozik, a keret pedig újra meg újra átküldésre kerül addig, amíg egyszer sikeresen át nem jut a csatornán.

Nevezzük „keretidőnek" azt az időtartamot, amely egy szabványos, fix hosszúságú keret átviteléhez szükséges (azaz a keret hosszát osztva a bitsebességgel)! Tegyük fel, hogy a felhasználók végtelen populációja a kereteket Poisson-eloszlás szerint állítja elő, keretidőnként átlagosan N keretet. (A végtelen populáció feltételezése azért szükséges, hogy a felhasználók blokkolódása esetén se csökkenjen N értéke.) Ha $N > 1$, akkor a felhasználói közösség nagyobb intenzitással állítja elő a kereteket, mint ahogyan azt a csatorna kezelni képes, így majdnem minden keret ütközést fog elszenvedni. Elfogadható áteresztőképesség $0 < N < 1$ tartományban alakulhat ki.

Az új keretek elküldése mellett az állomásoknak a régi, ütközést szenvedett keretek újraindítását is el kell végezni. Tegyük még fel, hogy keretidőnként k számú új és régi keretegyüttes elküldési kísérleteinek valószínűsége ugyancsak Poisson-eloszlású, és keretidőnkénti középértéke G. Világos, hogy $G \geq N$. Kis terhelés esetén (vagyis $N \approx 0$) csak kevés ütközés fordul elő, így az újraadások száma is kicsi, tehát $G \approx N$. Nagy terheléskor sok az ütközés, így $G > N$. Bármekkora is a terhelés, az S áteresztőképességet mindig a G aktuális terhelés, és a sikeres átvitel valószínűségének (P_0) szorzata

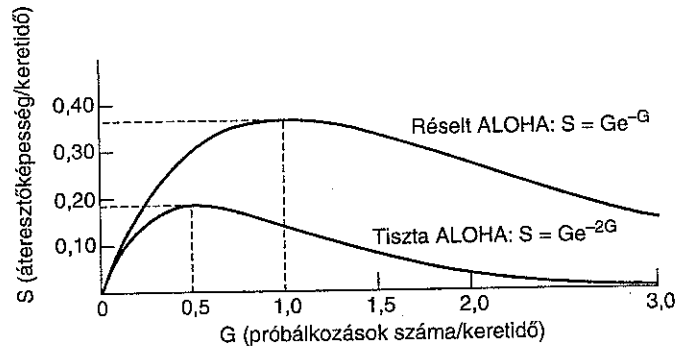


4.2. ábra. *A sötétített keret ütközésveszélyes szakasza*

adja meg. Tehát $S = GP_0$, ahol P_0 annak a valószínűsége, hogy egy keret nem szenved ütközést az átvitel során.

Egy keret akkor nem szenved ütközést, ha elküldésének első pillanatától kezdve egy keretideig nem próbálkozik más állomás keretküldéssel. Ezt szemlélteti a 4.2. ábra. Milyen körülmények között érkezhet meg sértetlenül az ábrán besötétített keret? Legyen t az egy keret elküldéséhez szükséges idő. Ha $t0$ és $t0 + t$ időpontok között valamelyik felhasználó keretet küldött, akkor annak vége ütközni fog a sötétített keret elejével. Igazság szerint, a sötétített keret sorsa már azelőtt megpecsételődött, mielőtt az első bitjét elküldték volna, de mivel az egyszerű ALOHA rendszerben az állomások nem figyelik a csatornát az adás megkezdése előtt, nem tudhatják, hogy egy keret már úton van. Hasonlóan, azok a keretek, amelyek küldését a $t0 + t$ és $t0 + 2t$ intervallumban kezdik meg, a sötétített keret végével fognak ütközni.

Annak valószínűsége, hogy egy adott keretidő alatt k új keret keletkezik a feltételezett Poisson-eloszlás szerint:



4.3. ábra. *Az ALOHA rendszerek áteresztőképessége a forgalmi igény függvényében*

$$P_r[k] = \frac{G^k e^{-6}}{k!} \qquad (4.2.)$$

így annak valószínűsége, hogy nem keletkezik keret e^{-G}. Két keret hosszúságú időintervallumban a keletkezett keretek várható száma $2G$. Annak valószínűsége tehát, hogy a teljes kritikus szakasz során nem lesz egyéb forgalom, $P_0 = e^{-2G}$. Az $S = GP_0$ egyenlőséget felhasználva:

$$S = Ge^{-2G}$$

Az áteresztőképesség és az aktuális forgalmi igény kapcsolatát a 4.3. ábra szemlélteti. Az áteresztőképesség G = 0,5-nél éri el a maximumát, az $1/(2e)$ értéket, amely megközelítőleg 0,184. Ez annyit jelent, hogy az elérhető legjobb csatornakihasználtság legfeljebb 18%-os lehet. Ez az eredmény nem túl biztató, de egy olyan módszerrel szemben, amelyben mindenki akkor adhat, amikor csak akar, aligha várható el 100%-os kihasználtság.

Réselt ALOHA

1972-ben Roberts olyan módszert publikált, amellyel egy ALOHA rendszer kapacitása megduplázható. Javaslata szerint az időt diszkrét szeletekre kell osztani, amelyek hossza a keretidőkhöz igazodik. Az eljárás megköveteli viszont, hogy a felhasználók megegyezzenek az időintervallumok határainak pontos helyében. A szinkronizálás egy lehetséges módja, hogy egy kijelölt állomás, akárcsak egy óra, speciális jelet bocsát ki az időintervallumok kezdetén.

Roberts módszerében, amely **réselt ALOHA (slotted ALOHA)** néven lett ismert, Abramson **egyszerű ALOHA (pure ALOHA)** rendszerével ellentétben, a terminálok nem kezdhetnek el adni bármikor, amikor leütik a kocsi-vissza billentyűt, hanem meg kell előbb várniuk a következő időrés kezdetét. Ezáltal a folyamatos egyszerű ALOHA diszkrétté alakul. Mivel a keretek kritikus szakasza a felére csökken, annak valószínűsége, hogy a tesztkeretünkkel azonos időrésben ne legyen egyéb forgalom e^{-G}, ami az

$$S = Ge^{-G} \qquad (4.3.)$$

egyenlőséghez vezet. Ahogy a 4.3. ábrán láthatjuk, a réselt ALOHA G = 1-nél éri el áteresztőképességének maximumát $S = 1/e$, azaz megközelítőleg 0,368 értékkel, amely kétszer akkora, mint az egyszerű ALOHA esetén. Ha a rendszer G = 1-gyel működik, akkor egy üres időrés előfordulási valószínűsége (4.2.)-ből következően 0,368. A réselt ALOHA-val elérhető legjobb kihasználtság mellett az időrések 37%-a üres, 37%-a sikeres, és 26%-a ütközéses lesz. Magasabb G értékek mellett az üres időrések száma ugyan csökken, viszont exponenciálisan megnő az ütközéses időrések száma. A növekedés ütemének szemléltetése céljából végezzünk el egy egyszerű számítást egy tesztkeret elküldésével kapcsolatban. Annak valószínűsége, hogy elkerüli az ütközést, megegyezik annak valószínűségével, hogy mások nem adnak az adott időrésben, vagyis e^{-G}. Az ütközés valószínűsége tehát $1 - e^{-G}$. Pontosan k kísérletet követelő átvitel valószínűsége (vagyis a sikeres átvitelt $k - 1$ ütközés előzte meg):

$$P_k = e^{-G}(1 - e^{-G})^{k-1}$$

Ebből következik, hogy a kocsi-vissza billentyű leütése után az átviteli kísérletek várható E száma:

$$E = \sum_{k=1}^{\infty} kP_k = \sum_{k=1}^{\infty} ke^{-G}\left(1 - e^{G}\right)^{k-1} = e^{G}$$

E-nek G-től való exponenciális függése azt eredményezi, hogy a csatorna terhelésének kis növekedése is drasztikusan csökkentheti annak teljesítményét.

4.2.2. Vivőjel-érzékeléses többszörös hozzáférésű protokollok

Réselt ALOHA használatával az elérhető legjobb csatornakihasználtság $1/e$. Ez meglepően jó eredmény, hiszen egy olyan rendszerben, ahol az állomások tetszés szerint, a többi állomás tevékenységének figyelembevétele nélkül adhatnak, sok ütközésnek kell bekövetkeznie. Helyi hálózatokban azonban az állomások érzékelhetik más állomások tevékenységét, így viselkedésüket azokhoz igazíthatják. Ennek köszönhetően ezek a hálózatok $1/e$-nél sokkal jobb csatornakihasználást érhetnek el. A következőkben néhány olyan protokollt mutatunk be, amelyek megnövelik a csatorna teljesítőképességét.

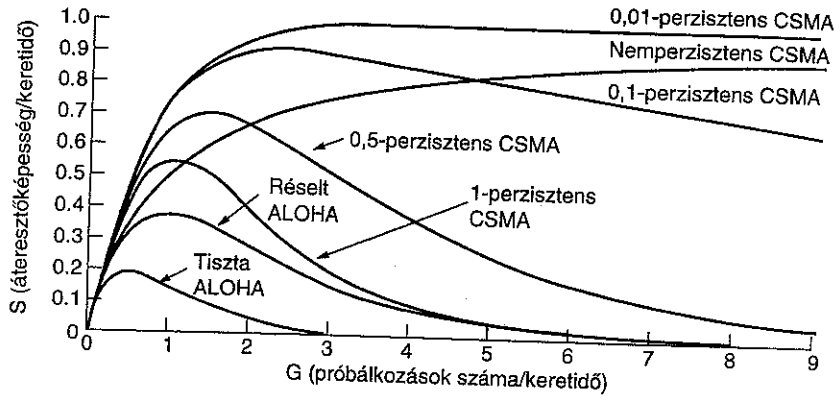
Azokat a protokollokat, amelyekben az állomások figyelik a csatornán folyó forgalmat, és ennek megfelelően cselekszenek, **csatornafigyelő protokolloknak** vagy vivőjel-érzékeléses protokollnak **(carrier sense protocols)** nevezik. Ezek közül többet a gyakorlatban is használnak. Kleinrock és Tobagi (1975) több ilyen protokollt is részletesen elemzett. Az alábbiakban a csatornafigyelő protokollok több verzióját is bemutatjuk.

Perzisztens és nemperzisztens CSMA

Az első csatornafigyelő protokoll az **1-perzisztens CSMA (Carrier Sense Multiple Access – vivőjel-érzékeléses többszörös hozzáférés).** Amikor egy állomás adni készül, először belehallgat a csatornába, hogy eldönthesse, használja-e azt éppen egy másik állomás. Ha a csatorna foglalt, akkor addig vár, amíg az ismét szabad nem lesz. Amikor az állomás szabad csatornát érzékel, elküld egy keretet. Ha ütközés következik be, akkor az állomás véletlen hosszúságú ideig vár, majd újból elölről kezdi a keret elküldését. A protokollt 1-perzisztensnek nevezik, mivel a várakozó állomás 1 valószínűséggel adni kezd, amint üresnek érzékeli a csatornát.

A protokoll teljesítményét nagymértékben befolyásolja a terjedési késleltetés. Csak kis esélye van annak, hogy miután egy állomás adni kezdett, egy másik állomás, amelyik éppen adásra kész állapotba került, érzékelje a csatorna foglaltságát. Ha az első állomás által küldött jel nem éri el a másodikat, akkor az szabadnak érzékelve a csatornát szintén adni kezd, így ütközés áll elő. Minél nagyobb a terjedési késleltetés, annál fontosabbá válik ez a jelenség, és annál rosszabb lesz a protokoll teljesítőképessége.

Ütközések még akkor is előfordulnának, ha a terjedési idő nulla lenne. Ha két állomás is adásra kész állapotba kerülne miközben egy harmadik állomás használja a csatornát, mindketten udvariasan kivárnák, amíg a csatorna szabaddá válik, majd egyszerre kezdenének adni, ami ütközést eredményezne. Ha türelmesebbek lennének, kevesebb ütközés keletkezne. Ez a protokoll azonban még így is sokkal jobb az egyszerű ALOHA-nál, mivel mindkét állomás elég illedelmes ahhoz, hogy elkerülje az ütközést a harmadik állomás keretével. Ebből intuitív módon következik, hogy a módszer nagyobb teljesítményt képes elérni, mint az egyszerű ALOHA. Pontosan ugyanez igaz a réselt ALOHA-ra is.



4.4. ábra. *Véletlen hozzáférésű protokollok összehasonlítása a terhelés függvényében mért csatornakihasználtság alapján*

A második csatornafigyelő protokoll a **nemperzisztens CSMA (nonpersistent CSMA)**. Ebben a protokollban tudatosan arra törekedtek, hogy az állomások ne legyenek mohóak. Küldés előtt az állomás megfigyeli a csatornát. Ha senki sem forgalmaz, akkor az állomás elkezdhet adni. Ha azonban foglalt a csatorna, nem folytatja folyamatosan a megfigyelést, hogy a forgalom megszűntével azonnal megkezdje az adást, hanem véletlen hosszúságú ideig várakozik, és ekkor elölről kezdi az algoritmust. Ismét intuitívan következtetve, ennek a módszernek jobb kihasználtsághoz, de nagyobb késleltetésekhez kell vezetnie, mint az 1-perzisztens CSMA-nak.

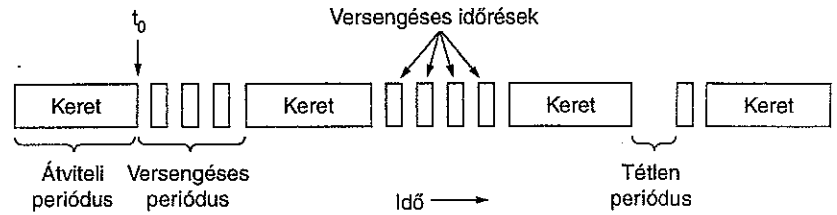
Az utolsó a **p-perzisztens CSMA (p-persistent CSMA)** protokoll. Réselt csatornát alkalmaz, és a következőképpen működik. Amikor egy állomás adásra kész állapotba kerül, megvizsgálja a csatornát. Ha az szabad, akkor p valószínűséggel forgalmazni kezd, vagy $q = 1 - p$ valószínűséggel visszalép szándékától a következő időrésig. Ha a következő időrésben a csatorna még mindig szabad, akkor ismét p, illetve q valószínűséggel ad vagy visszalép. Ez a folyamat addig folytatódik, amíg a keret elküldésre nem kerül, vagy egy másik állomás forgalmazni nem kezd. Az utóbbi eset ugyanolyan hatású, mintha ütközés következett volna be (azaz véletlen hosszúságú ideig vár, majd újból elölről kezdi az algoritmust). Ha az állomás már kezdetben érzékeli a csatorna foglaltságát, akkor vár a következő időrésig, és csak ott kezdi a fent leírt algoritmust. A 4.4. ábra nem csak e három protokoll, hanem az egyszerű és réselt ALOHA protokollok áteresztőképességét is szemlélteti a forgalmi igények függvényében.

CSMA ütközésérzékeléssel

A perzisztens és nemperzisztens protokollok egyértelmű előrelépést jelentenek az ALOHA rendszerhez képest, hiszen az állomások nem kezdenek el adni, ha a csatornát foglaltnak érzékelik. További fejlődés, hogy az állomások az ütközés érzékelése után azonnal befejezik adásukat. Másképpen fogalmazva, ha két állomás tétlennek érzékelve a csatornát egyszerre kezd adni, majd érzékelik az ütközést, akkor nem fejezik be a már visszavonhatatlanul sérült keretek csatornára küldését, hanem az ütközés érzékelését követően azonnal felfüggesztik tevékenységüket. A sérült keretek küldésének megszakítása időt és sávszélességet takarít meg. Ezt a protokollt **CSMA/CD-nek (Carrier Sense Multiple Access with Collision Detection – ütközésérzékeléses CSMA)** nevezik, és elterjedten használják LAN-ok MAC protokolljaként.

A CSMA/CD más LAN protokollokhoz hasonlóan a 4.5. ábrán látható koncepcionális modellt használja. A t_0-val jelölt ponton egy állomás befejezi keretének küldését. Ezen a ponton minden olyan állomás, amelyik kész kerettel rendelkezik, megkísérelheti azt elküldeni. Ha kettő vagy több állomás is kezd el egyszerre adni, akkor ütközés jön létre. Az ütközés könnyen észlelhető, ha az állomások megfigyelik a csatorna feszültségszintjét, vagy a rajta megjelenő impulzushosszat, majd azt összehasonlítják a kibocsátott jel paramétereivel.

Ha egy állomás ütközést érzékel, megszakítja a forgalmazást, véletlen hosszúságú ideig vár, majd feltételezve, hogy időközben egyetlen állomás sem kezdett adásba, megkezdi a keretének továbbítását. Következésképpen CSMA/CD modellünkben versengéses és átviteli periódusok váltják egymást, melyek között tétlen állapotok is megjelenhetnek, amikor egyik állomás sem forgalmaz (pl. nincs továbbítandó adat).



4.5. ábra. *A CSMA/CD mindig a következő állapotok közül az egyikben lehet: versengési, átviteli vagy tétlen*

Most pedig ismerkedjünk meg közelebbről a versengés algoritmusának részleteivel! Tegyük fel, hogy két állomás pontosan ugyanabban a t_0 időpillanatban kezd el adni. Mennyi idő telik el, amíg észlelik az ütközést? Ennek a kérdésnek a megválaszolása döntő jelentőségű a versengési periódus hosszának meghatározásához, és ebből következően a késleltetés és az áteresztőképesség kiszámításához. Az ütközés érzékeléséhez szükséges minimális idő az, ami alatt a jel az egyik állomástól a másikig eljut.

Ezen az alapon gondolkozva azt hihetnénk, hogy egy állomás, amely a kábel teljes terjedési idejének megfelelő ideig nem észlel ütközést, biztos lehet abban, hogy megszerezte a kábel használati jogát. A „megszerezte" kifejezésen azt értjük, hogy az összes többi állomás tud a folyó átvitelről, és így nem zavarják azt meg. Ez a következtetés azonban hibás! Tekintsük a következő legrosszabb eshetőséget! Jelöljük a két legtávolabb levő állomás között a jel terjedési idejét τ-val! A t_0 időpillanatban az egyik állomás elkezd adni. A $\tau - \varepsilon$ időpillanatban, tehát mielőtt a jel megérkezhetett volna a legtávolabbi állomáshoz, az is forgalmazni kezd. Természetesen szinte azonnal észreveszi az ütközést és leáll, de az ütközés által okozott kis zaj nem jut vissza az első állomáshoz $2\tau - \varepsilon$ időn belül. Vagyis legrosszabb esetben egy állomás csak akkor lehet biztos abban, hogy megszerezte a csatornát, ha már 2τ ideje forgalmaz ütközés nélkül. Ezért a versengési intervallumot úgy modellezzük, mintha egy 2τ réshosszúságú réselt ALOHA rendszer lenne. Egy 1 km hosszúságú koaxiális kábelen $\tau \approx 5\mu s$. Az egyszerűség kedvéért feltételezzük, hogy minden rés csak egy bitet tartalmaz. Ha már egy állomás megszerezte a csatornát, akkor tetszőleges sebességgel adhat, és nemcsak 1 bit/2τ-val.

Fontos észrevennünk, hogy az ütközésellenőrzés *analóg* folyamat. Az állomás hardverének figyelnie kell a csatornát adás közben. Ha mást olvas vissza, mint amit a csatornára kiírt, akkor tudja, hogy ütközés történt. Ezért a kódolásnak olyannak kell lennie, hogy az ütközést ilyen módon fel lehessen ismerni (pl. két 0 V-os jel ütközését elég nehéz lenne így észrevenni). Emiatt általában speciális kódolást szoktak használni.

A CSMA/CD fontos protokoll. A fejezet során később még részletezzük egy változatát, az IEEE 802.3-at (Ethernet), amely nemzetközi szabvány.

A félreértések elkerülése végett érdemes megjegyezni, hogy semmilyen MAC protokoll nem garantál biztonságos kézbesítést. Még ütközések nélkül is előfordulhat, hogy a vevő valamilyen oknál fogva rosszul másolja le a keretet (pl. szabad pufferterület hiánya, elvételt megszakítás).

4.2.3. Ütközésmentes protokollok

Bár a CSMA/CD protokolloknál nincs ütközés, miután egy állomás már megszerezte a jogot a csatorna használatára, de sajnos annál több fordulhat elő a versengési periódus alatt. Ezek az ütközések nagymértékben befolyásolják a rendszer teljesítményét, különösen akkor, ha a kábel hosszú (vagyis τ nagy) és rövidek a keretek. A nagyméretű, nagy sávszélességű optikai hálózatok terjedésével a nagy τ és kis keretméret kombináció problémája egyre égetőbbé válik. Ez a fejezet olyan protokollokat mutat be, amelyek ütközés nélkül teszik lehetővé a csatornáért folytatott verseny lefolytatását még a versengési periódus folyamán is.

A protokollok bemutatásánál feltételezzük, hogy N állomás van, és hogy ezek az állomások „beégetett", 0-tól $N-1$-ig terjedő saját címmel rendelkeznek. Nem okoz problémát, ha néhány állomás az idő egy részében inaktív. Az alapvető kérdés az marad, melyik állomás használhatja a csatornát egy-egy sikeres átvitel befejeztével. Továbbra is a 4.5. ábrán bemutatott diszkrét versengési időrésekkel dolgozó modellt fogjuk használni.

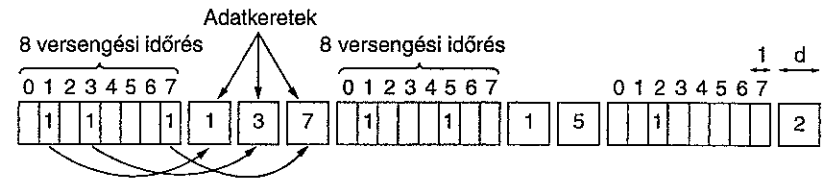
Egy bit-térkép protokoll

Az első bemutatásra kerülő ütközésmentes protokollban, az **alapvető bit-térkép eljárásban (basic bit-map protocol)** az ütköztetési periódus pontosan N időrésből áll. Ha a 0-s állomás adni szeretne, akkor 1-es bitet küld a 0-s (első) versengési időrésben. Ez alatt az időrés alatt másik állomások nem használhatják a csatornát. A 0-s állomástól függetlenül, az 1-es állomásnak szintén megvan a lehetősége, hogy az 1-es (második) időrés jelzőbitjét 1-re állítsa, ha van kész kerete. Általánosan a j-edik állomás a j időrésben jelezheti egy 1-es bittel, ha van elküldésre váró kerete. Az N darab időrés elküldése után, mindegyik állomás pontosan tudja, hogy mely állomások szeretnének forgalmazni. Ekkor számsorrendben megkezdhetik a tényleges adattovábbítást, mint ahogy azt a 4.6. ábra is szemlélteti.

Mivel az állomások megegyeztek, hogy mikor melyikük következik soron, sohasem jöhet létre ütközés. Miután az utolsó adni kívánó állomás is elküldte a keretét – ezt az eseményt mindegyik állomás könnyedén észlelheti –, egy újabb N időréses versengési periódus veheti kezdetét. Ha egy állomás szerencsétlenül pont akkor lesz adásra kész, miután már kitöltötte a számára fenntartott versengési időrést, akkor kénytelen az aktuális körben csendben maradni és megvárni, amíg a versengési periódus számára fenntartott időrése ismét körbeér. Azokat a protokollokat, amelyekben – ehhez hasonlóan – a forgalmazási igényt a tényleges adatátvitel előtt kell adatszórással (broadcast) jelezni, **foglalásos protokollnak (reservation protocol)** nevezik.

Röviden vizsgáljuk meg e protokoll teljesítményét. Az egyszerűség kedvéért az időt versengési időrésekben mérjük, és egy-egy adatkeretet d hosszúságúnak tételezünk fel. Ha a terhelés kicsi, akkor továbbítandó adatkeretek hiányában a versengési bit-térkép fog újra meg újra ismétlődni a csatornán.

Nézzük a helyzetet egy alacsony azonosítóval (pl.: 0 vagy 1) rendelkező állomás szemszögéből. Amikor küldésre kész állapotba kerül, az „aktuális" rés általában valahol a bit-térkép közepén fog járni. Átlagosan tehát $N/2$ időrést kell egy ilyen állomás-



4.6. ábra. *Az alapvető bit-térkép protokoll*

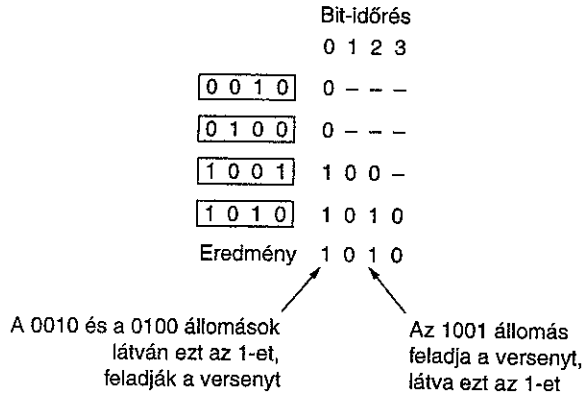
nak várnia az aktuális kör végéig, majd N időrést, amíg a következő versengési periódus befejeztével megkezdheti a forgalmazást.

A magasabb sorszámú állomások helyzete már kedvezőbb. Ezeknek általában a versengési periódus felét ($N/2$ időrést) kell csak várniuk az adás megkezdése előtt. Ezeknek az állomásoknak csak nagyon ritkán kell kivárniuk a következő versengési periódust. Mivel az alacsony sorszámú állomásoknak átlagosan $1{,}5N$ időrést, a magas sorszámúaknak pedig $0{,}5N$ időrést kell várniuk, az összes állomásra számított átlagos várakozási idő N időrés lesz. Alacsony terhelés mellett tehát egyszerűen számítható a csatorna hatékonysága. Keretenként a d adatbitre N többletbit (overhead) jut, így a hatékonyság $d/(N+d)$.

Nagy terhelés esetén, amikor mindegyik állomásnak mindig van küldenivalója, az N hosszúságú versengési periódus N adatkeret között oszlik meg, így minden keretre csak 1 többletbit adódik. Ebből a csatorna hatékonyságára $d/(d+1)$ adódik. Egy keret átlagos késleltetése két részből tevődik össze: az adott állomáson belüli sorbanállási késleltetésből, valamint további $N(d+1)/2$ időrésnyi késleltetésből, amelyet a belső sor elejére kerülve fog várni a tényleges elküldésig.

Bináris visszaszámlálás

Az alapvető bit-térképes protokoll legnagyobb hátránya az, hogy a versengési periódus hossza állomásonként 1 bittel nő. Bináris állomáscímeket használva azonban jobb eredményeket érhetünk el. Ez esetben a forgalmazni kívánó állomás elkezdi a bináris címét, a legnagyobb helyi értékű bittel kezdve, mindenkinek szétküldeni. Az összes állomás címének azonos hosszúságúnak kell lennie. Az elküldött címek azonos helyi értékű bitjei logikai VAGY kapcsolatba lépnek egymással. Ezt a protokollt **bináris visszaszámlálásnak (binary countdown)** nevezzük. A Datakit (Fraser, 1987) is ilyen protokollt használ.



4.7. ábra. *A bináris visszaszámlálás protokoll. A „–" azt jelzi, hogy az állomás nem forgalmaz*

A konfliktusok elkerülése érdekében szükség van egy kiegészítő szabályra is: amint egy állomás észleli, hogy 1-gyel lett felülírva egy olyan, magasabb helyi értékű címbit pozíció, ahol a saját címében 0 van, fel kell adnia a próbálkozást. Például, ha a 0010, 0100, 1001 és 1010 címekkel rendelkező állomások szeretnék használni a csatornát, akkor mindannyian elkezdik szétküldeni a legmagasabb címbitjüket, jelen esetben 0-t, 0-t, 1-et, illetve 1-et. Ezek logikai VAGY kapcsolata 1-et eredményez. A 0010 és a 0100 című állomások, látván az 1-et, feladják a versenyt, mivel látják, hogy a versenyben magasabb című állomás is részt vesz. Az 1001 és 1010 állomások tovább folytatják a versengést.

A következő bit 0, így mindkét állomás versenyben marad. Az ezt követő bit azonban 1, így az 1001 című állomás feladja a versengést. A győztes tehát az 1010 állomás lesz, mivel övé a legnagyobb cím. Miután megnyerte a „licitálást", továbbíthat egy keretet, amely után újabb verseny kezdődik a forgalmazás jogáért. A protokoll működését a 4.7. ábra illusztrálja.

A csatornakihasználtság ilyen protokoll mellett $d/(d+\log_2 N)$. Abban az esetben viszont, ha okosan választjuk meg a keretek felépítését, és az első mező pont a küldő címét tartalmazza, még ez a $\log_2 N$ bit sem veszik kárba, így a csatorna kihasználtsága 100% is lehet!

Mok és Ward (1979) kidolgozott egy olyan változatot a bináris visszaszámlálás protokollra, amely a soros felület helyett párhuzamos felületet használ. Ők javasolták a virtuális állomáscímek használatát is, amelyek lehetővé teszik, hogy minden sikeres átvitel után ciklikusan permutálva az állomások címét, azok az állomások is magasabb prioritáshoz juthassanak, amelyek már régóta hallgatni voltak kénytelenek. Például, ha a C, H, D, A, G, B, E, F állomások sorra a 7, 6, 5, 4, 3, 2, 1 és 0 prioritásokkal rendelkeznek, majd a D állomásnak sikerül továbbítania egy keretet, akkor a lista végére kerülve kialakul az új C, H, A, G, B, E, F, D prioritási sorrend. A C állomásnak tehát megmarad a 7 sorszáma, de az A állomás 4-ről 5-re léphet előre, míg a D állomás 5-ről 0-ra kénytelen visszalépni. A következő ciklusban tehát a D állomás csak akkor forgalmazhat, ha a többi állomás nem igényli a csatornát.

4.2.4. Korlátozott versenyes protokollok

Eddig két alapvető csatornamegszerzési stratégiát tárgyaltunk kábeles hálózatok esetén: a versenyhelyzetes (mint amilyen a CSMA) és az ütközésmentes módszereket. Mindegyik stratégiát megítélhetjük két fontos teljesítménymérő szám, a kis terhelés mellett fellépő késleltetés, illetve a nagy terhelés mellett fennálló csatornakihasználtság alapján. Kis terhelés esetén a versenyhelyzetes módszerek (azaz az egyszerű és a réselt ALOHA) a kedvezőbbek kis késleltetésük miatt. Ahogy nő a terhelés, a versenyhelyzetes protokollok egyre kevésbé vonzóak, mivel a csatorna megszerzésével eltöltött idő egyre növekszik. Az ütközésmentes protokollokra ennek éppen az ellenkezője igaz. Kis terhelés mellett nagy a késleltetésük, de ahogy a terhelés növekszik, a csatorna kihasználtsága egyre javul, ellentétben a versenyhelyzetes protokollokkal, ahol az ilyenkor egyre romlik.

Nyilvánvaló, hogy szerencsés lenne ötvözni a versenyhelyzetes és ütközésmentes

protokollok legjobb tulajdonságait, és olyan új protokollt tervezni, amely kis terhelés esetén versenyhelyzetes technikát használna a kis késleltetés érdekében, illetve nagy terhelés mellett ütközésmentes technikát alkalmazva a csatorna jó kihasználása érdekében. Ilyen, **korlátozott versenyes (limited contention protocol)** protokollok már léteznek, és ezekkel fejezzük be a csatornafigyelő protokollok tanulmányozását.

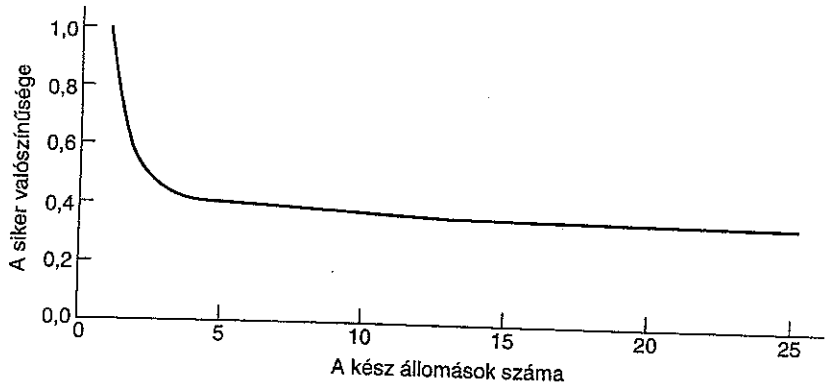
Az összes eddig tanulmányozott versenyhelyzetes protokoll szimmetrikus volt, vagyis az állomások p valószínűséggel próbálták megszerezni a csatornát, ahol a p minden állomásra azonos értékű volt. Érdekes, hogy a teljes rendszer teljesítményének növelése érdekében néha elég, ha olyan protokollt használunk, amely az állomásokhoz különböző valószínűségeket rendel.

Mielőtt áttérnénk az aszimmetrikus protokollok vizsgálatára, röviden tekintsük át a szimmetrikus protokollok teljesítményviszonyait. Ha a csatorna megszerzéséért egy adott résben k állomás verseng, és mindegyik p valószínűséggel adhat, akkor annak valószínűsége, hogy egy állomás megszerzi a csatornát $kp(1-p)^{k-1}$. Az optimális p megtalálása érdekében deriváljuk a kifejezést p szerint, az eredményt nullával egyenlővé tesszük, majd megoldjuk p-re az egyenlőséget. Ezt végrehajtva, p-re az $1/k$ értéket kapjuk, amelyet ha behelyettesítünk az eredeti kifejezésbe, a következő összefüggést kapjuk:

$$\Pr[\text{siker optimális } p \text{ mellett}] = \ldots \left(\frac{k-1}{k}\right)^{k-1} \tag{4.4}$$

A valószínűség függvény grafikonját a 4.8. ábrán láthatjuk. Kis állomásszám mellett a csatorna megszerzésének valószínűsége jó, de már öt állomás esetén is az esélyek az aszimptotikus $1/e$ érték közelébe zuhannak le.

A 4.8. ábra alapján nyilvánvaló, hogy egy állomás csatornamegszerzési esélyeit növelni csak a versenyhelyzetek számának csökkentésével lehet. A korlátozott versenyes protokoll pontosan ezt teszi. Először is az állomásokat (nem feltétlenül diszjunkt) cso-



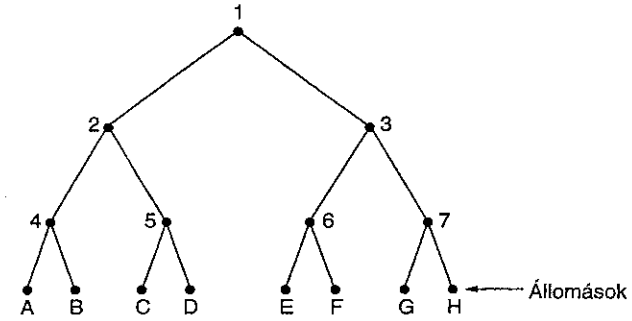
4.8. ábra. *A csatorna megszerzésének valószínűsége szimmetrikus versenyhelyzetes protokoll esetén*

portokra osztják. A 0-s résért csak a 0-s csoport tagjai versenghetnek. Ha valamelyikük megszerzi a csatornát, akkor elküldi keretét. Ha viszont ütközés fordult elő, vagy a rés kihasználatlan maradt, akkor máris az 1-es rés, és abban az 1-es csoport tagjai következnek és így tovább. Az állomások megfelelő csoportokra osztásával a résekre jutó versenyhelyzetek száma csökkenthető, így a rések a 4.8. ábra bal oldalához hasonló karakterisztikával működhetnek.

A trükk abban van, ahogyan az állomásokat a résekhez rendeljük. Mielőtt az általános esetet megvizsgálnánk, nézzünk meg néhány speciális esetet! Az egyik szélső eset az, amikor mindegyik csoport csak egy állomást tartalmaz. Az ilyen eset biztosítja, hogy ne legyen ütközés, hiszen résenként legfeljebb csak egy állomás versenghet a csatornáért. Ilyen protokollt már láttunk korábban (pl. bináris visszaszámlálás). A következő speciális esetben csoportonként két állmás van. Annak esélye, hogy ezek egy résben egyszerre akarjanak adni p^2, amely kis p-kre elhanyagolható. Az azonos réshez tartozó állomások számának növelésével nő az ütközések valószínűsége, viszont a bittérkép mérete csökken anélkül, hogy állomások elvesztenék adási lehetőségüket. A határeset az, amikor az összes állomás egy csoportba kerül, ez a réselt ALOHA. Olyan dinamikus állomás-hozzárendelésre lenne szükségünk, amely kis terhelés esetén egy réshez sok, míg nagy terhelés esetén pedig csak néhány (esetleg csak egy) állomást rendelne.

Adaptív fabejárási protokoll

A hozzárendelés egyik legegyszerűbb módja az, ahogy a II. világháború alatt az amerikai hadseregben a katonák szifiliszes fertőzöttségét vizsgálták (Dorfman, 1943). Röviden ezt úgy végezték, hogy a hadsereg N katonájától vért vettek, amelyek mindegyikéből egy kis részt egyetlen kémcsőbe öntöttek, és ezt vizsgálták antitestek után kutatva. Ha nem találtak, akkor az összes abba a csoportba tartozó katona egészséges volt. Ha azonban találtak antitestet, akkor az N katonát két csoportba osztották, és a vérmintákból két újabb keveréket állítottak elő. A folyamatot addig ismételték, amíg meg nem találták a fertőzött katonát.



4.9. ábra. *Nyolc állomásból álló fa*

Az algoritmus számítógépes változatához (Capetanakis, 1979) kényelmes, ha az állomásokat a 4.9. ábrának megfelelően egy bináris fa leveleinek képzeljük el. Egy sikeres keretátvitelt követően az első versengési résben, a 0-s résben, az összes állomás szabadon versenghet a csatorna megszerzéséért. Ha az egyik állomásnak sikerül, akkor minden rendben van. Ha ütközés következik be, akkor az 1-es résben már csak a 2. csomópont alatti részfa állomásai versenyezhetnek. Ha az egyikük megszerzi a csatornát, akkor a keretét követő rés a 3. csomópont alatti állomások számára lesz fenntartva. Ha viszont két vagy több 2. csomópont alatti állomás is forgalmazni szeretett volna az 1-es résben, akkor a bekövetkező újabb ütközés miatt a 2-es résben a 4. csomópont alatti részfa állomásai következhetnek.

Lényegében tehát, ha a 0-s időrés alatt ütközés következik be, akkor a küldésre kész állomások felkutatása érdekében megkezdődik a fa mélységi bejárása. A rések a fa egyes csomópontjaihoz vannak rendelve. Ha ütközés következik be, akkor a keresés rekurzívan az adott csomópont bal és jobb gyermekcsomópontjánál folytatódik. Ha egy bitrés kihasználatlan marad, vagy csak pontosan egy állomás küld benne, akkor annak a csomópontnak a keresése befejeződik, hiszen alatta nincs több küldésre kész állomás. (Ha ugyanis több is lett volna, akkor ütközésnek kellett volna bekövetkeznie.)

Amikor a rendszer terhelése nagy, aligha éri meg a 0-s rést az 1. csomóponthoz rendelni, hiszen csak abban a valószínűtlen esetben nem következne be ekkor ütközés, ha legfeljebb csak egyetlen állomás rendelkezne küldésre kész kerettel. Hasonló megfontolásból lehetne vitatkozni a 2. és 3. csomópont átugrásán is. A kérdést általánosan fogalmazva: melyik szinten érdemes elkezdeni a keresést? Világos, hogy minél nagyobb a terhelés, a fában annál mélyebben érdemes kezdeni a keresést. Tegyük fel, hogy az állomásoknak q értékű jó becslése van arra, hogy hány kész állomás van éppen (pl. az addigi forgalom megfigyeléséből következtetve).

A fa gyökerétől számozzuk meg az egyes szinteket! A 4.9. ábrán az 1. csomópont a 0-s szint, a 2. és 3. csomópont az 1-es szint stb. Vegyük észre, hogy az i-edik szint minden csomópontjához az alatta levő állomások 2^{-i} része tartozik! Ha a q adásra kész állomás a fában egyenletesen elosztva helyezkedik el, akkor egy i. szinten levő csomópont alatt várhatóan $2^{-i}q$ darab van. Intuitív módon arra következtethetünk, hogy a keresést azon a szinten optimális elkezdeni, ahol a résenként versengő állomások száma átlagosan 1, azaz $2^{-i}q = 1$. Az egyenletet megoldva az $i = \log_2 q$ értéket kapjuk.

Bertsekas és Gallager (1992) az alapalgoritmus számos továbbfejlesztett változatát állította elő és vizsgálta meg. Például tekintsük azt az esetet, amikor csak a G és H állomás akar adni. Az 1-es csomópontnál ütközés következik be, így a 2-esre kerül a sor, amikor üres marad a csatorna. Teljesen felesleges lenne a 3-as csomópont vizsgálata, hiszen biztos, hogy ütközés fog bekövetkezni, mivel tudjuk, hogy az 1-es csomópont alatt kettő vagy több kész állomás is van, viszont a 2-es alatti részfában egy sincs, így ezeknek mind a 3-as alatt kell lenniük. A 3-as csomópont vizsgálata kimarad, és helyette a 6-os következik. Mikor kiderül, hogy ez alatt a csomópont alatt sincs adásra kész állomás, kihagyható a 7-es tesztelése, és azonnal a G állomásra kerülhet a sor.

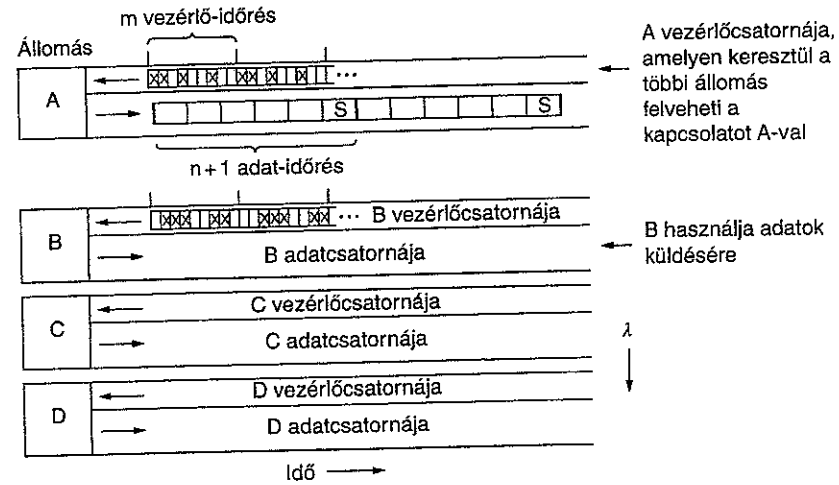
4.2.5. Hullámhosszosztásos többszörös hozzáférésű protokollok

Egy másfajta megközelítése a csatorna-hozzárendelésnek, hogy az eredeti csatornát FDM, TDM vagy mindkettő alkalmazásával alcsatornákra bontják, és ezeket osztják ki dinamikusan az igényeknek megfelelően. Ehhez hasonló módszert közkedvelten használnak üvegszálas LAN-ok esetén, amikor a különböző, egy időben zajló párbeszédekhez különböző, egyedi hullámhosszakat (frekvenciákat) rendelnek. A következő részben egy ilyen protokollt fogunk megvizsgálni (Humblet és mások, 1992).

Egy teljesen üvegszálas LAN kiépítésének legegyszerűbb módja, ha egy passzív csillagkapcsolót használunk a 2.10. ábrának megfelelően. Tulajdonképpen, minden állomás felől két fényvezető szál csatlakozik egy üveghengerhez. Az egyik szál a henger felé kimenetként szolgál, míg a másik szál a henger felől szállítja az állomás bemeneti adatait. Az állomások kimeneti fénye megvilágítja az egész üveghengert, így az összes állomás képes azt érzékelni. A passzív csillagok akár több száz állomást is képesek kezelni.

Ahhoz, hogy egyszerre több átvitel is történhessen, a színspektrumot csatornákra (hullámhossztartományokra) kell osztani a 2.24. ábra szerint. Ebben, a **WDMA (Wavelength Division Multiple Access – hullámhosszosztásos többszörös hozzáférés)** néven ismert protokollban minden állomáshoz két csatornát rendelnek. Egy keskeny csatorna szolgál az állomás felé érkező vezérlőjelek átvitelére, míg egy szélesebb csatorna az állomás adatkereteinek továbbítására.

Minden csatorna a 4.10. ábrán szemléltetett módon időrések csoportjaira van osztva. Jelöljük a vezérlő csatorna időréseinek számát m-mel, az adatcsatorna időréseinek számát pedig $n + 1$-gyel, ahol n időrés tényleges adatforgalomra szolgál, míg az utolsót az állomás használja aktuális állapotának jelzésére (általában azt mutatja meg,



4.10. ábra. *Hullámhosszosztásos többszörös hozzáférés*

hogy a fenti csatornák mely időrései szabadok). Mindkét csatornán az időrések újra meg újra ismétlődnek. A 0-s időrést speciális jelzéssel látják el, hogy a később bekapcsolódók is észlelhessék. Az összes csatornát egy központi óra szinkronizálja.

A protokoll három forgalomosztályt támogat: (1) állandó adatsebességű, összeköttetés alapú forgalom – ilyet igényelnek a tömörítetlen videoadatok; (2) változó adatsebességű, összeköttetés alapú forgalom – ilyet igényel a fájlátvitel; (3) datagram forgalom – ilyet igényelnek az UDP csomagok. A két összeköttetés alapú protokoll esetén az alapötlet a következő: ha *A* kommunikálni akar *B*-vel, akkor először egy ÖSSZEKÖTTETÉS-KÉRÉS (connection request) keretet kell beszúrnia *B* vezérlőcsatornájának egyik szabad időrésébe. Ha *B* elfogadja, akkor megkezdődik a kommunikáció *A* adatcsatornáján keresztül.

Minden állomáson az alábbi két adó- és két vevőáramkör működik:

1. Egy fix hullámhosszon működő vevő, amely az állomás saját vezérlőcsatornáját figyeli.

2. Egy hangolható hullámhosszú adó, amely a többi állomások vezérlőcsatornáira dolgozik.

3. Egy fix hullámhosszon működő adó, amely az állomás adatkereteit állítja elő.

4. Egy hangolható hullámhosszú vevő, amellyel kiválasztható adatcsatornát figyelhet az állomás.

Másként fogalmazva, minden állomás a saját vezérlőcsatornáját figyeli a beérkező jelzéseket várva, de ráhangol az adó hullámhosszára az adatkeretek vételéhez. Hullámhossz hangolására Fabry–Perot vagy Mach–Zehnder interferométert használnak, amely a kívánt hullámhossz kivételével minden hullámhosszt kiszűr.

Nézzük azt az esetet, amikor *A* egy 2-es osztályú kommunikációs csatornát akar felépíteni a *B* állomással, mondjuk fájlátvitel céljából. Először *A* ráhangolja adatvevőjét *B* adatcsatornájára, és megvárja annak állapotjelző időrését, amelyből kiderül, hogy *B* vezérlőcsatornájának melyik rései foglaltak, és melyek szabadok. A 4.10. ábrán például láthatjuk, hogy *B* nyolc vezérlőrése közül a 0-s, a 4-es és az 5-ös szabad, a többi pedig foglalt, amit a keresztek jelölnek.

A kiválasztja az egyik szabad vezérlő-időrést, mondjuk a 4-est, és ebbe írja ÖSSZEKÖTTETÉS-KÉRÉS üzenetét. Mivel *B* állandóan figyeli a vezérlőcsatornáját, észleli a kérést, majd jóváhagyja azt azzal, hogy a 4-es időrést *A*-hoz rendeli. A hozzárendelés létrejöttéről mindenki tudomást szerezhet a vezérlőcsatorna állapotjelző időréséből. Amikor *A* látja a visszaigazolást, tudhatja, hogy létrejött az egyirányú kapcsolat. Ha *A* kétirányú kapcsolatot kért, akkor *B* megismétli ugyanezt az algoritmust *A* felé.

Elképzelhető, hogy amikor *A* megpróbálta felhasználni *B* 4-es vezérlőidőrését, egy *C* állomás pontosan ugyanezt teszi. Egyiküknek sem fog sikerülni az átvitel, és erről *B* állapotjelző időréséből szerezhetnek tudomást. Ekkor mindketten véletlenszerű ideig várakoznak, majd újra próbálkoznak.

Végre eljutottunk addig a pontig, hogy mindkét fél számára biztosított egy konfliktushelyzetektől mentes mód, amely segítségével rövid vezérlőüzeneteket küldhetnek egymásnak. A fájlátvitel lebonyolításához *A* küld *B*-nek egy üzenetet, amely például azt mondja: „Kérlek, figyeld az adatcsatornám 3. időrését, mert adatkeret van benne a számodra!" Amikor *B* megkapja a vezérlőüzenetet, áthangolja vevőjét *A* adatcsatornájára, hogy kiolvashassa az adatkeretet. A felsőbb réteg protokolljának megfelelően, *B* ugyanezzel a mechanizmussal küldhet nyugtázást vissza, ha kíván.

Figyeljünk fel arra, hogy konfliktus alakulhat ki, ha *A* és *C* egyaránt összeköttetésben áll *B*-vel, és egyszerre mindketten a 3. időrésben szeretnének adatkeretet küldeni! *B* véletlenszerűen választja ki az egyik keretet, a másik pedig elveszik.

Az állandó sebességű forgalomhoz a protokoll egy másik változatára van szükség. Amikor *A* elküldi vezérlőüzenetét, egyidejűleg elküldhet egy hasonló kérdést is: „Rendben van, ha minden 3. időrésben küldök keretet?" Ha *B* elfogadhatja ezt (korábban nem fogadott el ilyen kérést egyik állomástól sem a 3. időrésre), akkor garantált sávszélességű kapcsolat jön létre. Ha nem, akkor *A* egy másik ajánlattal próbálkozhat aszerint, hogy melyik kimeneti keretei szabadok.

A 3-as osztályú (datagram) forgalomhoz szintén egy újabb változatra van szükség. Az *A* állomás a szabadon talált (4-es) vezérlőkeretbe ÖSSZEKÖTTETÉS-KÉRÉS üzenet helyett, inkább egy ADAT A 3. IDŐRÉSBEN SZÁMODRA üzenetet ír. Ha *B* szabad a következő 3. időrésben, az átvitel sikeres, egyébként pedig a keret elveszik. Ilyen módon soha nincs szükség az összeköttetés kiépítésére.

A teljes protokoll különböző változatai lehetségesek. Például lehetséges, hogy nem rendelnek minden állomáshoz saját vezérlőcsatornát, hanem egyetlen közös vezérlőcsatornát használ az összes állomás. Minden állomás számára ki van osztva néhány időrés hasonlóan, mint amikor több virtuális csatornát nyalábolnak egyetlen fizikai csatornán.

Az is lehetséges, hogy minden állomásnak csak egyetlen hangolható adója, és egyetlen hangolható vevőáramköre van, az állomások csatornáját pedig m vezérlő, és $n + 1$ adat-időrésre osztjuk. A módszer hátránya az, hogy a küldő félnek hosszabb ideig kell várnia egy vezérlőidőrés megszerzésére, valamint az adatkeretek időben távolabb kerülnek egymástól, mivel közöttük további vezérlőinformáció is helyet kapott.

Számos egyéb WDMA protokollt terveztek már, amelyek csupán a részleteikben térnek el. Némelyek egyetlen vezérlőcsatornát használnak, mások többet. Néhány figyelembe veszi a továbbítási késleltetést, mások nem; néhány nyílt módon számol a hangoláshoz szükséges idővel, mások elhanyagolják azt. A protokollok különböznek összetettségükben, az átviteli képességeikben és a skálázhatóságukban. (Irodalom: Bogineni és mások, 1993; Chen és Yum, 1991; Chen 1994; Jia és Mukherjee, 1993; Levine és Akyildiz, 1995; valamint Williams és mások, 1993.)

4.2.6. Vezeték nélküli LAN protokollok

Ahogy egyre nő a hordozható számítógépek és kommunikációs eszközök száma, úgy nő az igény, hogy ezekről kapcsolatba lehessen lépni a külvilággal. Már az első hordozható telefonok is képesek voltak kapcsolatba lépni másik telefonokkal. Az első hordozható számítógépek még nem rendelkeztek ugyan ilyen képességekkel, nem

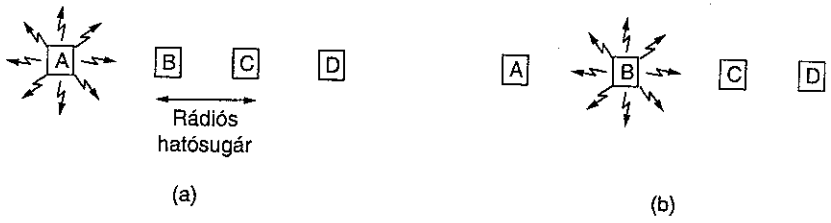
sokkal később azonban széleskörűen elterjedtek a modemek, így ha a hordozható gépek on-line kapcsolatot akarnak létesíteni, a fali telefoncsatlakozóhoz kell csatlakoztatni e gépeket. Mivel ez esetben vezetékes úton csatlakoztak egy fix hálózathoz, a számítógépek hordozhatóak voltak ugyan, de nem mobilak.

Az igazi mobilitás eléréséhez a hordozható gépeknek rádiós (vagy infravörös) jeleket kell használniuk a kommunikáció lebonyolításához. Ilyen módon az egyes felhasználók megírhatják és elolvashatják elektronikus leveleiket akár autóvezetés vagy csónakázás közben is. Egy olyan rendszer, amelyben hordozható számítógépek rádión kommunikálnak, már vezeték nélküli LAN-nak nevezhető. Ezek a hálózatok bizonyos mértékben különböző tulajdonságokkal rendelkeznek a szokványos LAN-okhoz képest, valamint speciális MAC alréteg protokollok használatát igénylik. További információhoz lehet jutni a vezeték nélküli LAN-okkal kapcsolatban a szakirodalomban (Davis és McGuffin, 1995; Nemzow, 1995).

Egy általános vezeték nélküli LAN egy irodaházból áll, amely köré bizonyos stratégia szerint bázisállomásokat telepítenek. A bázisállomásokat rézvezetékes, vagy üvegszálas hálózat köti össze. Ha a hordozható eszközök és a bázisállomások adóteljesítményét úgy állítják be, hogy 3 vagy 4 méteres hatósugaruk legyen, akkor minden szoba külön cellává, a teljes épület pedig egy celluláris rendszerré alakul éppen úgy, mint ahogyan azt a hagyományos celluláris telefonrendszerekről a 2. fejezetben tanultuk. A mobil telefonhálózatokkal szemben azonban itt minden cella csak egyetlen csatornával rendelkezik, amely lefedi a teljes rendelkezésre álló sávszélességet. Tipikusan ez a sávszélesség 1 és 2 Mb/s közé szokott esni.

Az alábbiakban azzal az egyszerűsítő feltételezéssel fogunk élni, hogy minden rádióadónak van egy fix hatótávolsága. Ha egy vevő két adónak is a hatósugarán belül tartózkodik, akkor a jelek összekeverednek, és használhatatlanná válnak (ez alól később lesz néhány kivétel). Fontos észrevenni, hogy van néhány vezeték nélküli LAN, amelyben nincs minden állomás a többi hatósugarán belül, és ez különböző komplikációkhoz vezet. Ezen kívül épületeken belül kialakított vezeték nélküli hálózatok esetén az állomások közötti falak nagymértékben befolyásolhatják az állomások adóinak effektív hatósugarát.

Vezeték nélküli LAN kialakításának egy naiv módja lehet a CSMA használata: füleljünk, és csak akkor kezdjünk el adni, ha nem észlelünk egyéb forgalmat. A probléma az, hogy ez a protokoll nem igazán használható, mivel az interferencia a vevőnél lép fel, és nem az adónál. Hogy megértsük a problémát, tekintsük a 4.11. ábrát, amelyen négy vezeték nélküli állomás látható. Számunkra most teljesen mindegy, hogy



4.11. ábra. *Egy vezeték nélküli LAN. (a) A ad. (b) B ad*

melyek a bázisállomások, és melyek a hordozható készülékek. A rádiós hatósugár akkora, hogy *A* és *B* egymás hatósugarán belül vannak, így egymás potenciális partnerei lehetnek. *C* szintén kapcsolatba léphet *B*-vel és *D*-vel, de *A*-val már nem.

Először vizsgáljuk meg, mi történik akkor, amikor *A* forgalmaz *B*-nek, ahogyan azt a 4.11.(a) ábra szemlélteti! Ha *C* belehallgat a csatornába, akkor nem hallhatja *A* adását, hiszen *A* kívül van a hatósugarán, így tévesen azt a következtetést vonhatja le, hogy elkezdhet sugározni. Ha *C* elkezd adni, akkor interferencia lép fel *B*-nél, és tönkreteszi az *A* által küldött keretet. Sokszor a **rejtett állomás problémájának (hidden station problem)** nevezik azt, amikor egy állomás nem képes érzékelni egy potenciális versenytársát, mivel az túl messze van tőle.

Vizsgáljuk most meg a fordított szituációt, amikor *B* ad *A*-nak, ahogy ez a 4.11.(b) ábrán látszik. Amikor *C* megvizsgálja a csatornát, hallva a folyamatban levő átvitelt, tévesen arra a következtetésre jut, hogy nem adhat *D*-nek, pedig ez az adás csak a *B* és *C* közötti tartományban tenné lehetetlenné a keretek vételét, ahol egyetlen címzett vevő sem található. Ezt a problémát sokszor a **megvilágított állomás problémája (exposed station problem)** néven emlegetik.

A probléma az, hogy az adás megkezdése előtt az állomás igazából arra lenne kíváncsi, hogy a vevő környezetében van-e aktivitás. A CSMA csupán azt képes megmondani, hogy a vivőérzékelést végző állomás környezetében van-e forgalom vagy sem. Vezeték esetén minden jel eljut minden állomáshoz, így az egész rendszerben minden pillanatban csak egyetlen átvitel történhet. Azonban a kis hatósugarú rádióhullámokat használó hálózatok esetében egyszerre több átvitel is lebonyolítható, ha azok célállomásai különbözőek, és kívül esnek egymás hatósugarán.

A probléma egy másik megközelítése az, ha elképzelünk egy nagy irodaépületet, amelyben minden dolgozónak van egy saját, vezeték nélküli hordozható számítógépe. Tegyük fel, hogy Linda szeretne egy üzenetet küldeni Miltonnak! Linda számítógépe megfigyeli a lokális környezetét, majd adásba kezd, mivel nem tapasztalt semmilyen egyéb aktivitást. Milton irodájában azonban így is előfordulhat ütközés, ha egy harmadik személy is üzenetet küld neki egy olyan pontról, amely Lindától túl távol van ahhoz, hogy számítógépe érzékelni tudja.

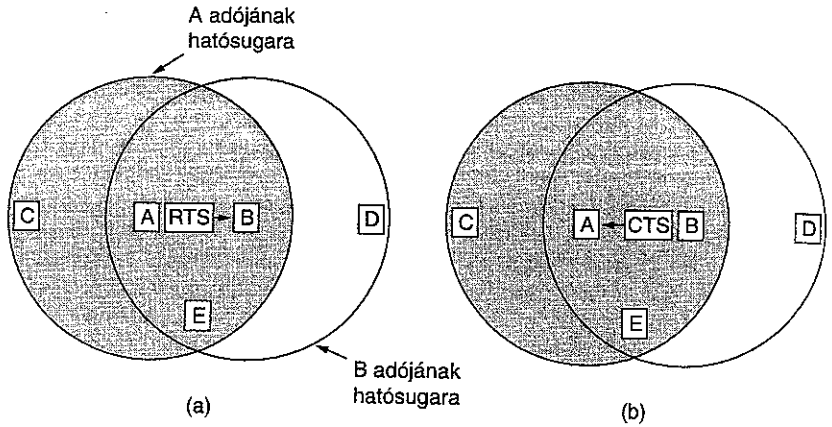
MACA és MACAW

Egy vezeték nélküli LAN-ok számára tervezett korai protokoll a **MACA (Multiple Access with Collision Avoidance – többszörös hozzáférés ütközések elkerülésével)** (Karn, 1990). Ez szolgált az IEEE 802.11, azaz a vezeték nélküli LAN-ok szabványának alapjául. A protokoll mögött rejlő alapötlet az, hogy az adónak rá kell vennie a vevőt, hogy adjon ki egy rövid keretet, amely következtében a hatósugarában tartózkodó állomások nem adnak a következő (hosszabb) adatkeret időtartama alatt. A MACA protokollt a 4.12. ábra szemlélteti.

Vizsgáljuk meg, hogyan küld *A* egy keretet *B*-nek. *A* azzal kezdi, hogy a 4.12.(a) ábrának megfelelően küld egy RTS (Request To Send – adási engedély kérése) keretet *B*-nek. Ez a rövid üzenet (mindössze 30 bájt) tartalmazza a soron következő adatkeret hosszát. Ekkor *B* a 4.12.(b) ábrán jelzett módon egy CTS (Clear To Send – adásra

kész) üzenettel válaszol. A CTS keret szintén tartalmazza az adatkeret hosszát (az RTS keretből másolja ki *B*). Amint megkapja a CTS keretet, *A* azonnal adni kezd.

Most nézzük meg, hogyan reagálnak azok az állomások, amelyek szintén fogják ezeknek a kereteknek valamelyikét! Azok az állomások, amelyek fogják az RTS keretet, közel vannak *A*-hoz, így legalább annyi ideig csendben kell maradniuk, amíg a CTS keret konfliktus nélkül visszaérkezhet az *A* állomáshoz. Azok az állomások, amelyekhez eljut a CTS keret, közel vannak a *B*-hez, így ezeknek csendben kell maradniuk az adatkeret átvitelének időtartama alatt, amelynek hosszát a CTS keretből deríthetik ki.



4.12. ábra. *A MACA protokoll. (a)* A RTS *üzenetet küld* B*-nek. (b)* B *egy* CTS *üzenettel válaszol* A*-nak*

A 4.12. ábrán *C* belül van *A* hatósugarán, de kívül esik *B* hatósugarán, így foghatja az RTS keretet *A*-tól, de nem hallhatja a *B*-től érkező CTS keretet. Mivel nem jutott el hozzá a CTS keret, szabadon forgalmazhat az adatkeret átvitelének időtartama alatt. Ezzel ellentétben a *D* állomás csak *B* hatósugarába esik bele, így nem foghatja az RTS, csak a CTS keretet. A CTS keretet megkapva értesül arról, hogy közel van ahhoz az állomáshoz, amelyik nemsokára egy adatkeretet szeretne fogadni, így nem forgalmazhat addig, amíg a keret küldése várhatóan be nem fejeződik. Az *E* állomás mindkét vezérlőüzenetet megkapja, így *D*-hez hasonlóan kénytelen csendben maradni az adatkeret továbbításának befejezéséig.

Az óvintézkedések ellenére is létrejöhet azonban ütközés. Például előfordulhat, hogy *B* és *C* egyszerre küld RTS üzenetet *A*-nak. Ezek ütközni fognak és elvesznek. Ha ütközés történt, akkor a sikertelen állomás (vagyis amelyik nem kapott CTS keretet válaszul a meghatározott időkorláton belül) véletlenszerű ideig várakozik, majd újból próbálkozik. A használt algoritmust kettes exponenciális visszalépésnek hívják, és bővebben fogjuk tárgyalni az IEEE 802.3 LAN-ok kapcsán.

Szimulációs eredményekre alapozva Bharghavan és munkatársai (1994) a teljesítőképesség növelése érdekében hangolták a MACA algoritmust, s az így keletkezett új protokollt **MACAW**-nak nevezték. Először is arra figyeltek fel, hogy az adatkapcsolati rétegben implementált visszajelzések hiányában az elveszett keretek újraküldése nem történik meg addig, amíg a szállítási réteg észre nem veszi azok hiányát, ami sokkal később következik csak be. A problémát úgy oldották meg, hogy bevezettek egy ACK keretet minden sikeresen továbbított adatkeret után. Észrevették azt is, hogy a CSMA rendelkezik egy hasznos képességgel – nevezetesen azzal, hogy egy állomás nem kezd RTS üzenet küldésébe addig, amíg észleli más állomások azonos célállomás irányába történő hasonló tevékenységét, így hát a protokollhoz adták a vivőérzékelést is. Elhatározták továbbá, hogy a visszalépéses algoritmust nem állomásonként, hanem adatfolyamonként (forrás-cél páronként) futtatják, ami a protokoll fair mivoltát növeli. Végül a rendszer teljesítményének növelése érdekében az állomásokhoz hozzáadtak egy mechanizmust, amellyel az állomások megoszthatják egymással torlódási információikat, valamint kidolgoztak egy módszert, amelynek köszönhetően a visszalépéses algoritmus kevésbé hevesen reagál az időszakos problémákra.

4.2.7. Digitális cellarádió

A vezeték nélküli hálózatok második megjelenési formája a digitális cellarádió, amely a 2. fejezetben tanult AMPS rendszer utódja. A digitális cellarádió kismértékben más környezetet biztosít, mint a vezeték nélküli LAN-ok, és így más protokollt használ. Tulajdonképpen ez inkább egy telefonáláshoz hasonló rendszer, amely nem milliszekundumokig, hanem percekig tartó összeköttetéseket feltételez, így a csatornakiosztást nem keretenként, hanem hívásonként érdemes megejteni. Ennek ellenére a módszerek ugyanolyan jók adatforgalom lebonyolítására is. Ebben a szakaszban a vezeték nélküli digitális rádiós rendszerek három teljesen különböző csatornafoglalási megközelítését fogjuk megtekinteni: GSM, CDPD és CSMA.

GSM – Global System for Mobile Communications

A celluláris telefonok első generációja, ahogyan az a 2. fejezetben szerepelt, analóg volt. A jelenlegi generáció már digitális, és rádiós csomagokat használ. A mobil kommunikációban a digitális átvitel számos előnnyel rendelkezik az analóggal szemben. Először is a hang-, adat- és faxszolgáltatások egyetlen rendszerbe integrálhatóak. Másodszor, ahogyan fejlődnek a beszédtömörítő algoritmusok, a csatornánként szükséges sávszélesség is egyre kisebb lesz. Harmadszor, az átvitel minőségének javítására hibajavító kódok használhatóak. Végül, a digitális jelek biztonsági okokból titkosíthatóak.

Annak ellenére, hogy nagyon szép lenne, ha az egész világ ugyanazt a digitális szabványt használná, sajnos ez nincs így. Az amerikai (IS–54) és a japán (JDC) rendszerek úgy lettek kialakítva, hogy illeszkedjenek az adott ország már létező analóg telefonhálózatához, így mindegyik AMPS csatorna használható akár analóg, akár digitális kommunikációra.

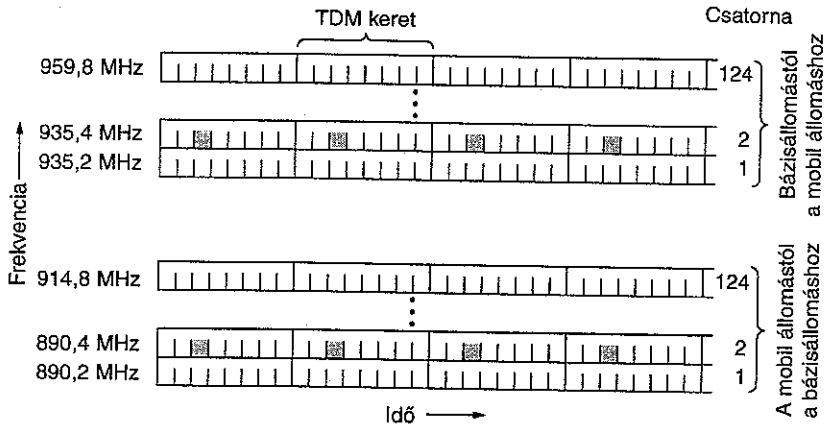
Ezzel ellentétben az európai digitális rendszer, a **GSM (Global System for Mobile Communications – mozgó objektumok kommunikációjára alkalmas globális**

rendszer) teljesen digitális rendszernek lett tervezve anélkül, hogy bármilyen kompromisszumot is kötöttek volna a régebbi rendszerekhez való illeszthetőség biztosítására (pl. már meglevő frekvenciasávok használata). Mivel az amerikaihoz képest a GSM az elterjedtebb rendszer (jelenleg Európán belül és kívül, több mint 50 ország használja), ezt fogjuk példaként használni a digitális cellarádiózás bemutatásához.

Eredetileg a GSM-et a 900 MHz-es sáv használatára tervezték. Később az 1800 MHz-es sávot is lefoglalták, és egy második rendszert alakítottak ki rajta, amely nagyon szoros kapcsolatban áll a GSM-mel. Az utóbbi rendszer neve **DCS 1800,** de alapvetően ez is GSM.

A teljes GSM szabványleírás több mint 5000 oldal hosszú. Az anyag jókora része a rendszer mérnöki vonatkozásaival foglalkozik, ezen belül főként a több útvonalas jelterjedést (multipath signal propagation) kezelni tudó vevőberendezések tervezésével, valamint az adók és vevők szinkronizálásának kérdéskörével.

A GSM rendszerek cellánként legfeljebb 200 duplex csatornával rendelkeznek. Mindegyik csatorna egy lefelé (a bázisállomástól a mobil állomás felé) irányuló forgalmat lehetővé tevő frekvenciát, és egy felfelé irányuló forgalmat lehetővé tevő frekvenciát (a mobilállomás felől a bázisállomáshoz) tartalmaz. A 4.13. ábrának megfelelően mindegyik frekvenciasáv 200 kHz széles.

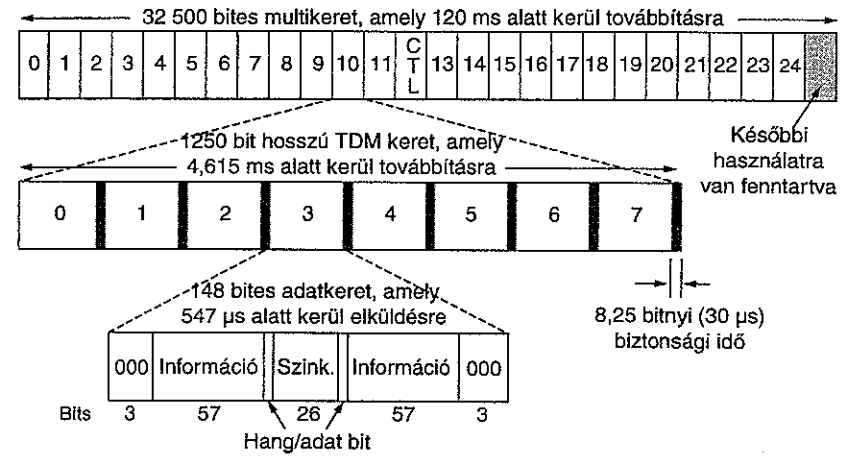


4.13. ábra. *GSM, amely 124 frekvenciacsatornát használ, amelyek közül mindegyik egy 8 időréses TDM rendszert használ*

A 124 frekvenciacsatorna közül mindegyik időosztásos nyalábolás használatával, egyidejűleg nyolc különböző összeköttetést támogat. Minden éppen aktív állomást hozzárendelik valamelyik csatorna egyik időréséhez. Ilyen módon elvileg 992 csatornát támogathat a rendszer cellánként, ám ezek közül sok nem használható, mert frekvenciakonfliktus lépne fel a szomszédos cellákkal. A 4.13. ábrán mindkét irányban négy darab besötétített időrés látható, amelyek mind ugyanahhoz a csatornához tartoznak. Ha a 890,4/935,4 MHz és 2-es időréshez rendelt mobilállomás adni szeretne a bázisállomás felé, akkor az alsó négy besötétített időrést (valamint az ezeket és e négy

rést követő további részeket) fogja igénybe venni az adatok elküldéséhez mindaddig, amíg mindet el nem küldte.

A 4.13. ábrán látható TDM időrések egy komplex keretezési hierarchia részét képezik. Mindegyik TDM résnek megvan a maga sajátságos felépítése, az időrések csoportjai pedig multikereteket formálnak, amelyeknek szintén jellegzetes a struktúrája. Ennek a hierarchiának egy egyszerűsített változata látható a 4.14. ábrán. Itt láthatjuk, hogy mindegyik TDM rés 148 bites adatkeretekből épül fel. Mindegyik adatkeret három 0 bittel kezdődik és végződik. Ezek a keretek elkülönítését segítik. Az adatkeretben található még két 57 bites *Információs* mező, amelyekhez tartozik egy vezérlőbit. Ez a bit jelzi, hogy a következő *Információs* mező hangot vagy adatokat tartalmaz. Az *Információs* mezők között található egy 26 bites *Szinkron* mező, amely segítségével a vevő az adó kerethatáraihoz szinkronizálódhat. Egy adatkeret elküldéséhez 547 μs-ra van szükség, de az adó csak 4,615 ms-onként küldhet egy-egy keretet, mivel hét másik állomással osztozik a csatornán. A csatornák teljes kapacitása 270,833 b/s, amely nyolc felhasználó között oszlik szét. Leszámítva a fejrészek átviteléhez szükséges sávszélességet, mindegyik összeköttetésen egy tömörített hangjel, vagy egy 9600 b/s-os adatjel vihető át.



4.14. ábra. *A GSM keretezési struktúra egy részlete*

Ahogy a 4.14. ábrán látható, nyolc adatkeret alkot egy TDM keretet, és 26 TDM keretből áll össze egy 120 ms hosszú multikeret. A multikeret 26 TDM keretének minden egyes keretében a 12-es rést vezérlésre használják, míg a 25-ös rést fenntartják későbbi használatra, így csak 24 rés használható a felhasználók forgalmának továbbítására.

A 4.14. ábrán bemutatott 26 rést tartalmazó multikeret mellett használnak még egy 51 rést tartalmazó multikeretet is (ez nincs az ábrán). Ezeknek a réseknek egy része olyan vezérlőcsatornákat tartalmaz, amelyeket a rendszer felügyeletére használnak. A **közérdekű vezérlőcsatorna (broadcast control channel)** egy, a bázisállomás által

generált folytonos adatfolyam, amely a bázisállomás azonosítóját és a csatorna állapotinformációját tartalmazza. Az összes mobilállomás figyeli ennek a csatornának a jelszintjét annak megállapítására, hogy mikor léptek át egy újabb cellába.

A **megkülönböztetett vezérlőcsatorna (dedicated control channel)** szolgál a helymeghatározás, a regisztráció, valamint a hívásfelépítés lebonyolítására. Gyakorlatilag minden bázisállomás kezel egy adatbázist, amelyben nyilvántartja az aktuálisan a fennhatósága alá eső mobilállomásokat. Az adatbázis karbantartásához szükséges információkat a megkülönböztetett vezérlőcsatornán továbbítják.

Végül van egy **közös vezérlőcsatorna (common control channel),** amely három logikai alcsatornából tevődik össze. Ezek közül az első a **felhívó csatorna (paging channel),** amelyen keresztül a bázisállomás jelezi a beérkező hívásokat. Az összes mozgó állomás folyamatosan figyeli ezt a csatornát, olyan hívások után kutatva, amelyekre válaszolniuk kell. A második a **véletlen hozzáférésű csatorna (random access channel),** amelyen réselt ALOHA rendszer szerint igényelhetnek az állomások időrést a megkülönböztetett vezérlőcsatornára, amely használatával aztán hívásfelépítést kezdeményezhetnek. Az így lefoglalt időrésekről értesítés a harmadik alcsatornán, a **hozzáférés engedélyező csatornán (access grant channel)** érkezik az állomások felé.

Mindent összevetve, a GSM egy igen összetett rendszer, amit alátámaszt az is, hogy a csatornaelérést a réselt ALOHA, az FDM és a TDM módszerek kombinációjával kezeli. A GSM rendszerről, és annak olyan aspektusairól, amelyekre itt nem tudtunk kitérni (pl. a protokoll rétegeinek felépítése) további információhoz Rahnema (1993) munkáiból juthatnak.

CDPD – Cellular Digital Packet Data

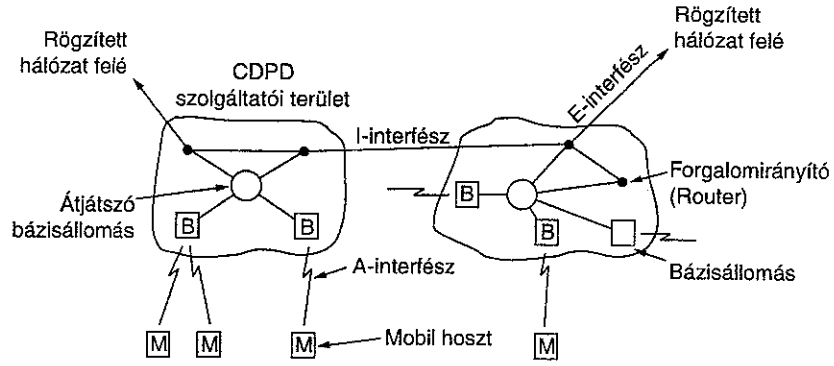
A GSM alapvetően vonalkapcsolt rendszer. Egy hordozható számítógép egy speciális modem segítségével pontosan ugyanúgy bonyolíthat le egy hívást, mintha a vezetékes telefonhálózatot használná. Az ilyen stratégia azonban rejteget problémákat. Gondot okozhat, hogy sűrűn fordul elő a bázisállomások közötti átváltás (handoff), amely akár még rögzített állomások esetén is előfordulhat (forgalomkiegyenlítési szempontok miatt sokszor átadhatják a felhasználókat egymásnak a bázisállomások). Minden váltás kb. 300 ms-nak megfelelő adatmennyiség elvesztésével jár. Másik probléma, hogy GSM rendszerekben nagy hibázási arányok is előfordulhatnak. Hamar fárasztóvá válhat, ha a begépelt „a" betű helyett „m" jelenik meg a másik oldalon. Végül, a vezeték nélküli hívások drágák, a díjazás pedig nem az elküldött bájtok számától, hanem a kapcsolat időtartamától függ, így a költségek hamar az egekbe szökhetnek.

Egy másfajta megoldás, amely képes ezeket a problémákat megoldani, a **CDPD (Cellular Digital Packet Data – celluláris datagram szolgálat).** Ez egy digitális, csomagkapcsolt datagram szolgáltatás, amely a 2. fejezetben részletezett AMPS rendszerre épül, és azzal teljesen kompatíbilis is marad. Az alapötlet az, hogy tetszőleges tétlen 30 kHz-es csatornát ideiglenesen meg lehet szerezni adatkeretek továbbítására, max. 19,2 kb/s sebességgel. A CDPD által megkövetelt nem kevés többletbit (overhead) miatt a hálózat tényleges adatsebessége inkább a 9600 b/s értékre tehető. Lévén egy összeköttetés nélküli, vezeték nélküli datagram rendszer, amely alkalmas például IP csomagok küldésére is a már meglevő celluláris telefonhálózaton keresztül, sok felhasználó számára érdekes lehetőséget jelent, így felhasználása robbanásszerűen növekszik.

A CDPD szorosan követi az OSI modellt. A fizikai réteg foglalkozik a modulációval és a rádiós átvitellel, amelyekkel itt nem foglalkozunk. Léteznek adatkapcsolati, hálózati, illetve szállítási protokolljai, de ezek sem igazán érdekesek számunkra. Ezek helyett először a rendszerről adunk egy általános leírást, majd megvizsgáljuk a közegelérési protokollt. A teljes CDPD rendszerről további információk gyűjthetők a szakirodalomból (Quick és Balachandran, 1993).

Egy CDPD rendszer három különböző állomásból épül fel: hordozható hosztokból, bázisállomásokból és átjátszó bázisállomásokból (base interface station). A CDPD szaknyelv ezeket hordozható végberendezésnek (mobile end system), mobil adatátviteli bázisrendszernek (mobile data base system), valamint mobil adatátviteli köztes rendszernek (mobile data intermediate system) nevezi. Ezek az állomások képesek kapcsolatba lépni rögzített hosztokkal, valamint szabványos routerekkel, mint amilyeneket a szokványos WAN-okban találhatunk. A hordozható hosztok a felhasználók hordozható számítógépeinek felelnek meg. A bázisállomások azok az adók, amelyek a mobil hosztokkal folytatnak párbeszédet, míg az átjátszó bázisállomások olyan speciális csomópontok, amelyek a CDPD szolgáltató adott területére eső bázisállomásokat összekötik egy szabványos (helyhez rögzített) routerrel, amelyen keresztül az adatforgalom átjuthat az Internetre, vagy tetszőleges WAN-ra. Ezt az elrendezést szemlélteti a 4.15. ábra.

Három interfészt definiál a CDPD szabvány. Az **E-interfész** a CDPD szolgáltatói területeket köti össze, a szolgáltató számára külvilágként megjelenő (external), helyhez rögzített külső hálózattal. Ezt a csatlakozási felületet nagyon jól kell definiálni, hogy a CDPD rendszereket minél több hálózathoz lehessen illeszteni. Az **I-interfészek** kötik össze a CDPD szolgáltatói területeket egymással, amelyek a szolgáltató szemszögéből belső kapcsolatokat jelentenek (internal). Ennek a felületnek szintén szabványosítottnak kell lennie, hogy a felhasználók szabadon átjárhassanak a szolgáltatói területek között. A harmadik a légi (air), vagyis az **A-interfész,** amely a bázisál-



4.15. ábra. *Egy példa a CDPD rendszerek felépítésére*