12. fejezet – Hibajelző kódok és Adatkapcsolati protokollok

Hibajelző kódok

Az előzőekben tárgyalt hibajavító kódokat jellemzően olyan átviteli közegekben célszerű használni, ahol a kapcsolat kevésbé megbízható, zajos így a hiba gyakran történik az átvitel során (pl.: vezeték nélküli kapcsolatok).
Abban az esetben viszont, amikor közvetlen réz- vagy optikai kábellel történik az adatátvitel – azaz az átviteli közeg, a csatorna megbízható – jellemzően kicsi hibaaránnyal találkozunk, így praktikusabb a hibás adategységeket újraküldeni. Felesleges a hosztok erőforrásait a kódolási és hibajavítási algoritmusokkal terhelni.

A legszélesebb körben használt – lineáris, szisztematikus blokk-kód rendszerű – három hibajelző kód a következő:

- Paritásbit képzése
- Ellenőrző összeg képzése (Checksum)
- CRC képzése (Cyclic Redundancy Check / Ciklikus Redundancia Ellenőrzés)

Paritásbit képzése

Hasonlóan a bitbeszúrás módszeréhez, ez esetben is egy bitet teszünk hozzá egy adategység, azaz kódszó végéhez. A paritásbit tehát egy „x” bitből álló kódszó „x+1”-edik bitje, azaz hibajelző bitje lesz. A paritásbit az ilyen módon kódolandó kódszó tartalma alapján generálódik, a kódszóban található 1-es bitek alapján. Az egyetlen szempont nem az 1-es bitek száma, hanem, hogy páros vagy páratlan darab van-e az adott kódszóban. Ebből logikusan következik, hogy megkülönböztetjük a páros vagy a páratlan paritás generálást. Páros paritás generálás esetén a paritásbit értéke 0, ha a kódszóban lévő 1-es bitek száma páros, és 1, ha ez a szám páratlan. Páratlan paritás generálás esetén a paritásbit akkor 0, ha a kódszóban lévő 1-es bitek száma páratlan, és 1, ha a számuk páros. Az előző két mondat első olvasatra logikátlannak tűnhet. A követendő logika ez esetben az, hogy az elnevezés a paritásbit generálása utáni állapotra utal. Miután a kódszóhoz hozzáadtuk a paritásbitet, az így előállított bitsorozatban található egyesek számára vonatkozik az elnevezés. [Ez a magyarázat a Fizikai Réteg anyagában is szerepel.]
A kódszó jellemzően 7 bit vagy 8 bit (azaz 1 bájt), de természetesen egyedileg is választható a hossza. A paritásbit egybites paritás esetén csak 1 bitnyi hibát tud biztonsággal jelezni, hiszen 1 hibás bit esetén az eredeti kódszó és a paritásbittel ellátott kódszó Hamming-távolsága 2. Ezért nem is célszerű túl hosszú kódszavakat használni, mivel ezzel csak a hiba esetén ismétlendő adatok méretét növeljük meg.

A tapasztalat szerint a megbízható csatornákban, ha hiba keletkezik, akkor az vagy csak 1 bitet érint, vagy több egymás utáni bitet. Nem jellemző hiba például a sorozatosan 1 bit jó, 1 bit hiba esete. A hiba általában valamilyen tranziens vagy konstans hatás következtében áll elő. A paritásbit tehát tökéletes védelem az egybites hibák jelzésére, de a több egymás utáni bithibát, azaz hibacsomót nem képes alaphelyzetben biztonsággal jelezni. Megoldás lehet erre az esetre kétdimenziós paritás.

	1. bit	2. bit	3. bit	4. bit	5. bit	6. bit	7. bit	Páratlan paritás	
1. átküldött bájt	1	1	0	1	1	0	0	1	→
2. átküldött bájt	1	0	1	1	0	1	0	1	→
3. átküldött bájt	1	0	0	0	1	1	0	0	→
4. átküldött bájt	0	0	1	1	0	0	1	0	→
5. átküldött bájt	0	1	1	1	1	1	1	1	→
6. átküldött bájt	1	1	0	0	0	1	0	0	→
7. átküldött bájt	1	0	1	0	0	1	1	1	→
Páratlan paritás	0	0	1	1	0	0	0	1	→
	↓	↓	↓	↓	↓	↓	↓	↓	paritás képzés

(A hibacsomó narancssárgával van jelölve)	1. bit	2. bit	3. bit	4. bit	5. bit	6. bit	7. bit	Páratlan paritás	
1. átküldött bájt	1	1	0	1	1	0	0	1	→
2. átküldött bájt	1	0	1	1	0	0	1	1	→
3. átküldött bájt	0	1	1	1	1	1	0	0	→
4. átküldött bájt	0	0	1	1	0	0	1	0	→
5. átküldött bájt	0	1	1	1	1	1	1	1	→
6. átküldött bájt	1	1	0	0	0	1	0	0	→
7. átküldött bájt	1	0	1	0	0	1	1	1	→
Páratlan paritás	**1**	**1**	**0**	**0**	0	**1**	**1**	1	→
	↓	↓	↓	↓	↓	↓	↓	↓	paritás képzés

Ellenőrző összeg (Checksum)

Már az elnevezés is utal arra, hogy itt „valami” összeadás állhat a képzés mögött. Maga az eljárás nem csupán az információ technológiában, és nem csak elektronikus környezetben ismert és használatos. Ellenőrző összeget tartalmaznak többek közt a vonalkódok, a személyi számok, adóazonosító számok és az EURO bankjegyek sorszámai is.

Ellenőrző összeget a legkülönfélébb módokon képezhetünk.

Például a 32 bites szavakból álló IPv4 csomagok is 16 bites ellenőrző összeget tartalmaznak. A módszer – ellentétben a paritásbit képzéssel – nem bitekkel, hanem bájtokkal (ez esetben 2 bájttal, azaz 16 bittel) dolgozik. Az ellenőrző összeg úgy képződik, hogy fejrészen belül az összes 32 bites szó két darab 16 bites számként (gyakorlatilag 4 jegyű hexadecimális számként) összeadásra kerül – kivéve magát az ellenőrző összeget. Az összeadás után keletkezett 5 jegyű hexadecimális szám legnagyobb helyiértékét – mint átvitelt – még hozzá kell adni az összeghez, amíg végül egy 4 jegyű hexadecimális értéket kapunk. Utolsó lépésként az így kapott összegnek az egyes komplemensét használjuk, azaz ez a bitsorrend kerül elküldésre, mint ellenőrző összeg.

Fixpontos bináris kódok (4 biten)

Egyenes (abszolútértékes)	
-7	1111
-6	1110
-5	1101
-4	1100
-3	1011
-2	1010
-1	1001
-0	1000
+0	0000
+1	0001
+2	0010
+3	0011
+4	0100
+5	0101
+6	0110
+7	0111

Kettes komplemens	
-8	1000
-7	1001
-6	1010
-5	1011
-4	1100
-3	1101
-2	1110
-1	1111
0	0000
+1	0001
+2	0010
+3	0011
+4	0100
+5	0101
+6	0110
+7	0111

Többletes	
-8	0000
-7	0001
-6	0010
-5	0011
-4	0100
-3	0101
-2	0110
-1	0111
0	1000
+1	1001
+2	1010
+3	1011
+4	1100
+5	1101
+6	1110
+7	1111

Egyes komplemens (Negációs)	
-7	1000
-6	1001
-5	1010
-4	1011
-3	1100
-2	1101
-1	1110
-0	1111
+0	0000
+1	0001
+2	0010
+3	0011
+4	0100
+5	0101
+6	0110
+7	0111

A Magyarországon használatos 10 jegyű magánszemélyeket azonosító adóazonosító első számjegye mindig a "8"-as szám, a 2-6. számjegyek az adott személy születési dátuma és 1867. január 1. között eltelt napok száma, a 7-9. számjegyek az azonos napon születettek megkülönböztetésére használt (pl. véletlenszerűen generált) számsorozat, a 10. számjegy pedig az ellenőrző összeg. Az ellenőrző összeg számítása a következő algoritmus szerint zajlik. Az adóazonosító jel első kilenc számjegyét egyenként megszorozzuk a számjegy pozíciójával, majd ezeket összeadjuk, az összeget elosztjuk 11-gyel, és ha az osztás maradéka kisebb tíznél, akkor ez lesz az ellenőrző összeg. Ha pedig tíz, akkor az eredmény nem használható adóazonosító jelként (változtatni kell a 7-9. számjegyeken – pl. adjunk hozzá egyet – és használható ellenőrző összeget kapunk).

Egy példa: 1967. február 13-án született személy adóazonosító jelének generálása:

1. az első számjegy fixen a "8"
2. a 2-6. számjegyek az 1967.02.13 és 1867.01.01 között eltelt napok számát tartalmazzák, ami ez esetben 36567
3. a 7-9. számjegyek legyenek most a lehető legkisebb érték (választhatnánk bármit 000 és 999 között ami csak ne 10 legyen az osztás utáni maradék), azaz 000
4. így az első kilenc számjegy: 836567000
5. az összeg képzése: 1x8 + 2x3 + 3x6 + 4x5 + 5x6 + 6x7 + 7x0 + 8x0 + 9x0 = 124
6. a 11-gyel való osztás maradéka (azaz az ellenőrző összeg maga): 3
7. a teljes adóazonosító jel így néz ki: 8365670003

Érdekes példa még a vasúti kocsi oldaláról leolvasható 12 jegyű szám (hivatalosan: pályaszám) érvényességéről gondoskodó ellenőrző összeg is. Az ellenőrző összeg képzése ez esetben úgy történik, hogy első lépésben a páros helyen álló számokat 1-el, a páratlan helyen álló számokat 2-vel kell megszorozni. Ezután a szorzatokban szereplő számjegyeket egyenként össze kell adni, majd az utolsó számjegyet a következő 10-el osztható számra kell kerekíteni, végül a felkerekített számból az eredeti összeget ki kell vonni.
51 55 39-51 001-X számsorozat esetében:

pályaszám:	5	1	5	5	3	9	5	1	0	0	1
szorzó:	2	1	2	1	2	1	2	1	2	1	2
szorzat:	10	1	10	5	6	9	10	1	0	0	2
összegezve:	1+0+	1+	1+0+	5+	6+	9+	1+0+	1+	0+	0+	2=	27

A 27 kerekítése után 30-at kapunk, a 30-ból a 27-et kivonva pedig megkapjuk az ellenőrző összeget (az X-el jelölt utolsó karaktert), vagyis ez esetben a 3-at.
A teljes pályaszám tehát 51 55 39-51 001-3.

CRC

Az eddigieknél erősebb rendszer, mivel a CRC a polinom kódok elvére épül, olyan polinomokra, melyekben csak 0-ás és 1-es együtthatók szerepelnek. A CRC kódolást W. Wesley Peterson dolgozta ki 1961-ben.
A CRC kiszámításához szükségünk van az üzenet polinomra „M”, valamint egy generátor polinomra „G”. Az üzenet polinomot a generátor polinommal elosztva kapjuk meg a maradékot „R”, azaz a redundanciát. Az adatátvitel ellenőrzésének az alapgondolata az, hogy ha egy polinomot elosztunk egy másik polinommal, és a maradékot kivonjuk az osztandóból, akkor az így kapott polinom maradék nélkül osztható lesz a korábbi osztóval. A generátor polinom lesz az osztó, értékét előzetesen az adó és a vevő oldalon is előre meg kell határozni, mert az átvitt keretben ez az információ nem szerepel. Mivel a generátor polinom nem szerepel az adatátvitelben, így lehetősége sincs megsérülni az adatátvitel bármely hibája esetén. Az IEEE802-ben használt generátor polinomok:

CRC-4	$x^4+x^1+x^0$	$1*x^4+0*x^3+0*x^2+1*x^1+1*x^0$	10011
CRC-8	$x^8+x^2+x^1+x^0$	$1*x^8+0*x^7+0*x^6+0*x^5+0*x^4+0*x^3+x^2+x^1+x^0$	100000111
CRC-10	$x^{10}+x^9+x^5+x^4+x^1+x^0$	*(fenti módon képezve a többi sorban is…)*	
CRC-12	$x^{12}+x^{11}+x^3+x^2+x^1+x^0$		
CRC-16	$x^{16}+x^{15}+x^2+x^0$		
CRC-CCITT	$x^{16}+x^{12}+x^5+x^0$		
CRC-32	$x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x^1+x^0$		

[CCITT – Comité Consultatif International Téléphonique et Télégraphique]

Az osztást a MODULO2 (MOD2) algebra szerint végezzük, azaz nem vesszük figyelembe az átvitelt. Az osztás esetében kivonások sorozatát kell alkalmaznunk, egészen addig, amíg a maradékunk kisebb nem lesz az osztónál. Polinomok esetében osztáskor az osztandó polinomnak nagyobbnak kell lennie, mint az osztó polinomnak. Az osztás során, ha az osztandó rövidebb (kevesebb bitet tartalmaz), akkor azt úgy értelmezzük, hogy „nincs meg benne”, és hozzá kell írni az osztandó következő bitjét.
A MODULO2 szabályai összeadáskor és kivonáskor az XOR művelettel egyeznek meg, szorzáskor pedig az AND művelettel.

MODULO2 összeadás		
A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	0

MODULO2 kivonás		
A	B	A - B
0	0	0
0	1	1
1	0	1
1	1	0

MODULO2 szorzás		
A	B	A x B
0	0	0
0	1	0
1	0	0
1	1	1

Ha a vételi oldalon az osztás nem maradék nélküli, akkor hiba van az adatátvitelben. Matematikailag könnyen belátható, hogy a CRC nem fog hibát jelezni, ha az eltérés az osztó többszöröse. Elegendően magas fokszámú osztó esetén ennek a valószínűsége viszont elhanyagolhatóan kicsi lehet.

Számítási példa CRC-re:
„M” = 1110010 azaz $x^6+x^5+x^4+x^1$
„G” = 10011 azaz $x^4+x^1+x^0$ (CRC-4)
A „G” polinom fokszáma esetünkben 4, ezért 4db 0-val bővítjük az „M” polinomot. Mivel „M” polinom vége 0-kat tartalmaz, ezért a kivonás valójában a maradék hozzáírását jelenti.

								kiegészítés			
"M" →	1	1	1	0	0	1	0	0	0	0	0
"G" →	1	0	0	1	1						
	0	1	1	1	1	**1**					
		1	0	0	1	1					
		0	1	1	0	0	**0**				
			1	0	0	1	1				
			0	1	0	1	1	**0**			
				1	0	0	1	1			
nem kivonható				0	0	1	0	1	**0**		
						1	0	1	0	**0**	
						1	0	0	1	1	
						0	0	1	1	1	**0**
kód →	**1**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**1**	**0**

Ellenőrzés:

kód →	1	1	1	0	0	1	0	1	1	1	0
"G" →	1	0	0	1	1						
	0	1	1	1	1	**1**					
		1	0	0	1	1					
		0	1	1	0	0	**0**				
			1	0	0	1	1				
			0	1	0	1	1	**1**			
				1	0	0	1	1			
nem kivonható				0	0	1	0	0	**1**		
						1	0	0	**1**	**1**	
						1	0	0	1	1	
maradék:						0	0	0	0	0	**0**

A számítás első ránézésre bonyolultnak tűnhet, de nagyobb polinomok esetén is hardverbe „huzalozott” módon számítható, azaz szoftveres erőforrások nélkül.
[CRC_számítás_v2.xls]
[CRC.exe]

Az adatkapcsolati protokollok néhány megvalósítási módja

- Korlátozás nélküli Simplex protokoll
- Simplex „megáll és vár” protokoll
- Simplex összetett protokoll
- Duplex protokoll
- Csúszóablakos protokoll
- Egybites csúszóablakos protokoll
- Visszalépés „n”-el technikájú protokoll
- Szelektív ismétlő protokoll

Korlátozás nélküli Simplex protokoll

Ez a létező legegyszerűbb egyirányú adatátviteli megvalósítás, de ebből fakadóan idealisztikus is, azaz elméletileg létezhet, de a gyakorlatban nem megvalósítható. Az adatátviteli sebesség, a feldolgozás nincs korlátozva: amilyen sebességgel küldi az adó a kereteket, a vevő ugyanilyen sebességgel képes ezt venni. Ez a gyakorlatban azt jelentené, hogy az adó és a vevő hálózati rétege mindig készen áll az adatcserére. A feldolgozási (kódolási, dekódolási) idő elhanyagolható, és a keretek esetleges tárolására szolgáló puffer kapacitás végtelen. Az adatkapcsolati rétegek közötti csatorna hibamentes, kerethiba, keretvesztés nem fordul elő.

Mindez azt jelenti, hogy sem forgalomszabályzásra, sem hibakezelésre sincs szükség, (hiszen mindez csak elvi síkon létezik). Közvetlen gyakorlati jelentősége nincs, de azért kerül megemlítésre, mert ennek segítségével hasonlíthatjuk össze a rideg valóságot és az idealisztikus elméleteket.

Simplex „megáll és vár” protokoll

A valóságban nagyon sok esetben a vevő nem képes olyan sebességgel feldolgozni a kereteket, ahogyan azt az adó küldeni képes. Valahogy az adót le kell lassítani, más szóval szinkronizálni kell, hogy a vevő küldött kereteket mindig fel tudja dolgozni. Ez csak egy módon lehetséges: informálni kell az adót arról, hogy mikor küldheti a következő keretet, azaz a vétel és a feldolgozás tényét nyugtázni kell. Vagyis a protokoll megköveteli az adótól, hogy egy keret elküldése után addig várjon, amíg egy nyugtakeret meg nem érkezik. Ezt a protokollt szokták “megáll és vár” (stop and wait) protokollnak nevezni.
Ez esetben is az adatforgalom maga Simplex, de a keretek két irányban áramlanak igaz, hogy csak különböző időpontokban, de ez már Half Duplex csatorna kialakítást igényli a fizikai réteg vonatkozásában. A protokoll jól működik az adatkeretek átvitelekor, hiszen a vevő csak akkor küld vissza nyugtát, ha a keret vétele helyes volt.
Hibamentes csatornán mindez kiválóan működhet. Mi van azonban akkor, ha vevő által küldött nyugtakeret sérül meg, vagy veszik el? Mivel ez esetben hibátlan nyugta nincs, az adó egy bizonyos idő múlva ismét elküldi a nem nyugtázott keretet, amit a vevő ismételten vesz. Így az ismételt keretben lévő adatok megkettőződve kerülhetnek a hálózati réteghez. Az adatok tényleges megkettőződést a keretek sorszámozásával küszöbölhetjük ki, de ezt a funkciót már egy másik protokoll nyújtja.

Simplex összetett protokoll

Ebben a protokollban már megoldott a keretek sorszámozása, azaz elkerülhető az ismételt keretek többszörös beépülése az adatfolyamba. Az adó egy számot helyez el minden elküldendő keret fejrészébe, és ezáltal a vevő eldöntheti, hogy először adott (illetve vett), vagy ismételt keretről van-e szó.

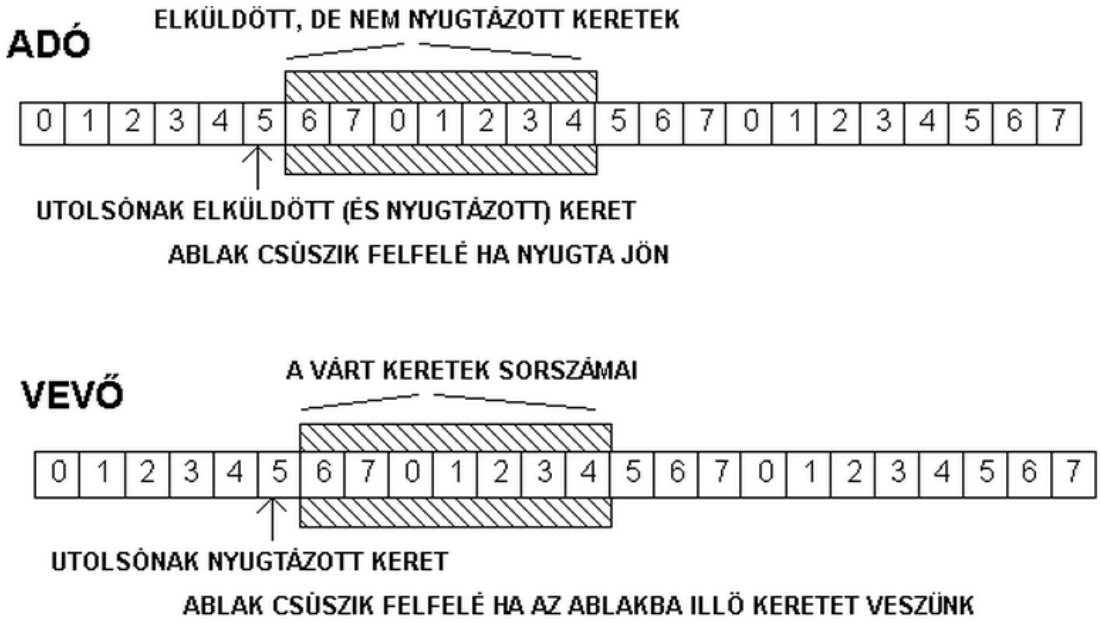
Duplex protokollok

Az előző esetekben az adatátvitel egyirányú volt, bár az utolsó két esetnél a nyugtázás miatt az ellenirányú átvitelre is szükség volt az adó informálása miatt. A gyakorlatban az adatátvitel is a legtöbbször kétirányú, ezért célszerű ezt a kialakítást is megvizsgálni. A megoldás lehetne két különálló, ellentétes irányú adatcsatorna használata, de az a

nyugtázás miatt valójában négy információs utat jelentene, ahol a nyugtacsatornák kihasználása kicsi lenne.
Jobb megoldás, ha mindkét irány számára ugyanazt a csatornát használjuk, hiszen az adatkereteket a nyugtakeretektől a keret fejrészében elhelyezett azonosítóval megkülönböztethető.

Csúszóablakos protokoll

Egy egyszerű megoldással az átviendő keretek számát csökkenthetjük, ha bármelyik irányba tartó adatkeretre ráültethetjük az előző ellenirányú adatkeret nyugtáját. Ezt szokták ráültetéses (Piggy-back) technikának is hívni. Hogy egy nyugta akkor is visszajusson, ha éppen nincs visszafelé küldött adatkeret, célszerű egy adott időzítés lejártakor a vevőnek önállóan útnak indítani a nyugtázó keretet. Persze, ha az adó eltérő időzítése miatt újra elküldi a keretet, akkor ez problémát jelent.
Ez a megoldás főleg akkor használható, ha mindkét oldal közel hasonló mennyiségű adatot akar a másik oldalhoz eljuttatni.
Az eddigiekben feltételeztük, hogy a csatornán mindig egy adatkeret, majd rá válaszul egy nyugtakeret halad. A valóságban a csatorna jobb kihasználását az teszi lehetővé, ha megengedjük, hogy a csatornán több keret is tartózkodhat. Ezt az eljárást csúszóablakos (Sliding Window) vagy forgóablakos protokolloknak nevezik.
A protokollban minden egyes kimenő keret egy 0 és 7 közötti sorszámot kap (ettől eltérő szám is lehet, de ez az általánosan használt érték). A lényeg az, hogy a sorban elküldendő keretek sorszámaiból az adó egy folyamatosan aktualizált listát kezel. A listában szereplő sorszámú keretek az adási ablakba (Sending Window) esnek.

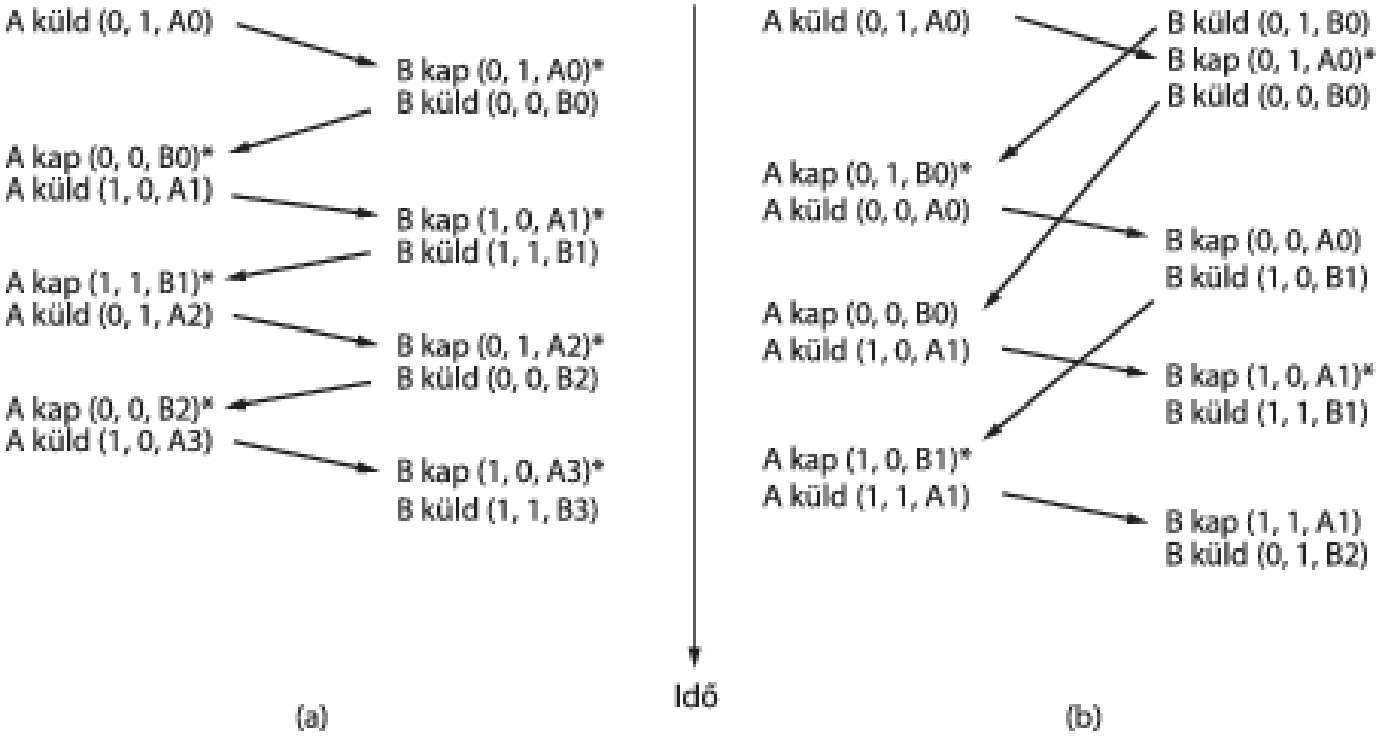


Az adó adási ablakában az elküldött, de még nem nyugtázott keretek vannak. Mikor egy nyugta megérkezik, az ablak alsó fele feljebb csúszik, lehetővé téve újabb keret elküldését. Nem kell a kereteket egyenként nyugtázni, ha pl. az ADÓ az 1-es sorszámú keretre kap nyugtát, ez azt jelenti, hogy nyugtázott a 6,7,0,1 keret. Mivel a kereteket esetleg újra kell adni, ezért az ablakban lévő kereteket ismételt adásra készen memória-pufferekben kell tartani. Az ADÓ ezenkívül az ablakban lévő minden keret elküldésétől eltelt időt nyilván kell hogy tartsa, és ha ez az időtúllépési értéknél (Timeout) nagyobb, akkor újra kell hogy küldje.
A vevő egy vételi ablakot (Receiving Window) tart fenn, amely az elfogadható keretek sorszámait tartalmazza. Ha bármelyik az ablakon kívüli keret érkezik, akkor eldobásra kerül. Ha az „n”-edik keret érkezik, akkor rá a nyugta a következő két feltétel együttes teljesülése esetén lesz visszaküldve.

1. Az „n”-edik keret még nem lett nyugtázva.
2. Minden keret az elsőnek várt és az „n”-edik között már megérkezett.

Egybites csúszóablakos protokoll

Ez a legegyszerűbb ilyen jellegű protokoll. Hasonló a „megáll-és-vár” protokollhoz, de az átvitel mindkét irányban folyik, és az ellenirányú keret hordozza az előzőleg küldött keret nyugtáját.
A példa kedvéért legyen a két állomás „A” és „B”. A keretek tartalma: (sorszám, nyugta, küldő betűjele és keretjelölése). Mivel mindig csak akkor lehet új keretet küldeni, ha nyugtázva van az előző, a sorszám és nyugta értéke is csak 0 vagy 1 lehet.
Az „A” kezdi az adást, küldi „B”-nek a keretet „Aküld(0,1,A0)”. [Az első keretben azért „1” a nyugta értéke, mert így „B” azt hiszi, hogy az előző (virtuális) küldés sikeres volt]. „B” veszi, és a nyugtát a saját keretével együtt visszaküldi. „Bkap(0,1,A0), Bküld(0,0,B0)”. „A” veszi „B” első keretét és saját előzőleg küldött kerete nyugtáját, majd küldi az újabb keretet „Akap(0,0,B0), Aküld(1,0,A1)”. És így tovább a lenti ábra (a) oldala szerint.
A protokoll nagyon jól működik: Ha például „A” nem kapja meg a nyugtáját, azaz „B” (0,1,B0)-át küld, akkor elküldi B-nek a (0,0,A0) keretet, amivel „A” nyugtázhatja a B0 keretet. Akár többször is küldheti (próbálkozhat), miközben „B” sorban adja a saját kereteit. A protokollt semmilyen elveszett keret, vagy a lejárt időzítés miatt újraküldött keret nem készteti arra, hogy kettőzött keretet adjon tovább a hálózati rétegnek, vagy egy keretet kihagyjon. Azonban keretkettőződés lép fel, ha „A” és „B” egyszerre kezd adni, hiszen ekkor induláskor 1-es nyugtával küldi el a saját keretét. Ez az amit el kell kerülni.



A fenti ábrán az (a) eset normális működést, a (b) eset pedig egy hibás működést ír le. A csillag azt jelzi, hogy az adatkapcsolati réteg továbbította a keretet a hálózati réteg felé.
Mind az adó mind a vevő számára egy elemes csúszóablak elegendő. Az adó az ablakba 0-át ír mikor elküld egy 0 sorszámú keretet, és amíg nem kap ezzel egyező nyugtát, újra küldi. Ha megjön a nyugta, akkor 1-et ír az ablakba, és várja a nyugtát. A vevő csúszóablaka kezdetben 0-át tartalmaz, azaz 0 sorszámot vár. Amennyiben ilyen keretet kap nyugtázza, és az ablakba 1-es sorszámot ír.

Visszalépés „n”-el technikájú protokoll

Ha a keretek átviteli ideje hosszú, például műholdas átvitel esetén, akkor nem szerencsés az a megoldás, hogy újabb keretet, csak az előző nyugtázása után indítunk, hiszen ezzel a csatorna kihasználtsága csökken – relatíve hosszú idő (néhány száz milliszekundum) is eltelhet adatforgalom nélkül. A megoldás az, hogy az adó nem 1, hanem „k” darab keretet küld el nyugtázás nélkül. Az „n”. keret elküldése után kezdi várni a nyugtákat és folytatni az „k+1”, „k+2”, stb... keretek küldését. Az ilyen esetben a csúszóablak mérete „k” kell hogy legyen. Ez a megoldás az úgynevezett csővonal (Pipelining). A név utal arra a szemléletes képre, hogy a keretek egy csőben haladnak, sorban egymás után.
Probléma azonban itt is adódhat. Mi van azonban akkor, ha egy keret a sorban megsérül? Két megközelítés ismert: az egyik a visszalépés „n”-el (Go back n) protokoll. Ennél a módszernél a vevő, a hibás keret utáni kereteket nyugtázatlanul eldobja, kényszerítve az adót az ismétlésre. Ez a stratégia 1-es méretű vételi ablaknak felel meg. Zajos vonalak

esetén ez a megoldás nagymértékben csökkenti az adatátviteli sebességet a sok újraküldés miatt, tehát pazarlóan, gazdaságtalanul bánik a sávszélességgel.

Szelektív ismétlő protokoll

A másik, csővonal esetén használható általános hibakezelési eljárás a szelektív ismétlés (Selective Repeat). Működésére jól utal a megoldás neve, miszerint ez esetben a hibás keretet követő összes jó keret tárolásra kerül, azaz csak a rossz keretek kerülnek eldobásra. Amikor az adó azt információt kapja, hogy volt hibás keret (az nem kap nyugtázást az adott keretről), akkor csak a hibás keretet küldi újra. Ennél a protokollnál, mind az adó mind a vevő fenntart ablakot a keretsorszámoknak. Az adó ablaka 0-tól az előre definiált maximális sorszámig növekszik, míg a vevő ablaka rögzített méretű, a megfelelő működés érdekében 1-nél mindenképpen nagyobb. Arról sem szabad megfeledkezni, hogy a nagyobb ablakméret nagyobb memóriát, több erőforrást igényel.
A szelektív ismétlő protokoll hatékonysága egy egyszerű módszerrel tovább javítható. A vevő, amikor hibás keretet észlel azonnal egy negatív nyugtát (Negative Ackowledgement) küld az adónak. A negatív nyugta még az időzítők lejárta előtt ki tudja kényszeríteni a keret újraküldését.