Segédlet – 2016

SZÁMÍTÓGÉP HÁLÓZATOK

Mérnökinformatika

Laczkó Sándor, Tiszai Tamás
ÓBUDAI EGYETEM NEUMANN JÁNOS INFORMATIKAI KAR

Tartalom

1. Fejezet

Számítógéphálózatok története.

A számítógépek közötti utasítások továbbítását kezdetben az emberek végezték. Elsőnek 1940-ben George Stibitz küldött utasításokat két telex-gép közötti hálózaton keresztül New York-on belül. A számítógépek kimeneti perifériáinak (telex-gépek) összekapcsolását először 1962-ben az ARPA (Advanced Research Projects Agency) keretében végezték. A kutatók 1964-ben Dartmouthban kifejlesztették az időosztásos rendszert, ami lehetővé tette egy nagy számítógép szolgáltatásainak nagyszámú felhasználó közötti megosztását. Ugyanebben az évben, az MIT, valamint a General Electric és Bell Labs fejlesztőibol álló csoport egy számítógépével (a DEC PDP-8-as) megvalósította egy telefonközpont vezérlését. Az első adatcsomagokat un. „datagramokat" továbbító hálózatot 1969-ben hozták létre. Amit „ARPANET"-nek neveztek és három amerikai egyetem gépei között valósítottak meg. Az ARPANET 50 Kbit/s-os hurok használatával működött. A hálózatok megjelenése nagyon nagy előrelépést jelentett az informatika témakörében. A hálózatok fejlődése során új összeköttetési lehetőségek jelentek meg. Az ARPANET utódjai közé a mai legnagyobb kiterjedésű hálózat az INTERNET is hozzátartozik. Napjainkban már nagyszámú hálózati típusról beszélünk, amelyeket többféle módon csoportosíthatunk is (területi kiterjedés, kompatibilitás, topológia, átviteli sebesség, átviteli módszer, kommunikáció iránya, kapcsolási technika, közeghozzáférési mód).

Az ARPANET-et később civil kezekbe adták, és NSF-Net[1] néven az egyetemek közötti kommunikációt szolgálta.

A Baud a telekommunikációban és az elektronika területén a „jelarány" mértéke, amely megmutatja, hogy egy adott átviteli média esetén hány modulált jelet továbbítottak 1 másodperc alatt. A névadó a francia Émile Baudot, a távírótechnikában használt Baudot-kód kidolgozója volt. 250 baud azt jelenti, hogy 250 jelet továbbítottak 1 másodperc alatt. Ha minden jel 4 bit információt hordozott, akkor egy másodperc alatt 1000 bitet vittek át. Ezt röviden 1000 bit/s-mal jelölhetjük.

A hálózatok típusai

1. PAN (**P**ersonal **A**rea **N**etwork) — Személyi hálózatok [0 … 10 m]
2. LAN (**L**ocal **A**rea **N**etwork) — Helyi hálózat [10 m … 10 km]
   - [Otthoni hálózat, Elektromos hálózaton keresztüli kommunikáció (Home-Plug)]
3. MAN (**M**etropolitan **A**rea **N**etwork) — Városi hálózat [10 km … 100 km]
4. WAN (**W**ide **A**rea **N**etwork) — Nagy kiterjedésű hálózat [100 km … ∞]

A hálózatok lehetnek **nyílt** (free), **zárt**, és **fizetős** hálózatok. Nyitott hálózatot egyre több helyen lehet érzékelni, nem is szükséges elemezni. Zárt hálózatokat többnyire vállalatok, cégek vagy kutatólaborok használnak. A fizetős hálózatok többnyire valamilyen szolgáltatás mellé vehetők igénybe, pl.: internetes kávézókban rendelés esetén kaphatjuk meg a bejelentkezésre az engedélyt.

[1] NSF-Net = National Science Foundation Network

A hálózatok csoportosíthatók az átviteli közeg jellege szerint is

1. Vezetékes hálózatok
2. Vezeték nélküli hálózatok

E két kategória aztán tovább bontható a vezeték típusa (csavart érpár, koaxiális kábel, üvegszál, stb.), illetve a vezeték nélküli átvitel módja (rádiós, mikrohullámú, lézeres, stb.) szerint.

Az OSI modell

A hálózatokkal kapcsolatos elméleti kérdések megoldására, valamint az egymással kompatibilis technológiák kidolgozásának megkönnyítésére szükség volt egy egységes fogalomrendszer kialakítására. Ebbe az irányba tett döntő lépést a *Nemzetközi Szabványügyi Szervezet – International* Standards Organization, ISO [valójában International Organization for Standardization, de ennek rövidítését nehéz kiejteni] – az 1970-es évek legvégén, amikor közreadta az ***Open Systems Interconnection*** – ***OSI*** – névre keresztelt ***ajánlást,*** amely olyan egységes alapelveket fogalmazott meg, amelyek nélkül a hálózatok fejlődése mára elképzelhetetlen lenne. A referencia-modell létrejötte megalapozta a későbbi szabványosítási törekvéseket, az egységes terminológia kialakításával jelentősen felgyorsította a számítógépes hálózatok fejlesztését.

Annak érdekében, hogy különböző gyártók különböző típusú berendezései együttműködhessenek, közösen elfogadott *szabványokra*, és/vagy ajánlásokra van szükségünk. Ha két gyártó eszközei azonos szabvány/ajánlás előírásait figyelembe véve készülnek, akkor ezek az eszközök – szerencsés esetben – képesek az együttműködésre. Amennyiben egy szabványt vagy ajánlást annak kidolgozói szabaddá tesznek – vagyis publikálják, és alkalmazásáért nem, vagy csak minimális licenc díjat kérnek – akkor *nyílt szabványról* beszélünk. Ha az alkalmazott eszközök együttműködése csupán ilyen nyílt szabványok betartását igényli, akkor az adott eszközt *nyílt rendszerű* berendezésnek tekintjük.

Manapság a számítógép hálózatokban döntően nyílt rendszerű berendezések üzemelnek. Ezen eszközök fejlesztésének elméleti alapjait az említett OSI referencia modell teremtette meg. Az ISO OSI ***referencia-modell*** értelmében két nyílt rendszer összekapcsolása során végrehajtandó feladatok összessége különféle ***rétegekbe*** – ***layer*** – csoportosítható. A modell értelmében az egymástól távol elhelyezkedő berendezések fizikailag valamely adatátviteli közegen keresztül kapcsolódnak, miközben e kapcsolat kialakításához és fenntartásához szükséges feladatokat a különféle rétegeket megvalósító áramkörök/programok végzik. A modell értelmében az egyes rétegek – kivéve a tényleges kapcsolatban álló 1. rétegeket – a két távoli nyílt rendszerben egymással csupán ***logikai*** kapcsolatban állnak, miközben tényleges adatátviteli igényeik lebonyolítására a közvetlenül alattuk elhelyezkedő réteg ***szolgáltatásait*** használják fel. E logikai kapcsolat során a rétegek szigorúan meghatározott szabályhalmaz – úgynevezett ***protokollok*** – segítségével érintkeznek (más szavakkal: az adott réteg egy ***protokollgépet*** képez). Miközben egy réteg logikai kapcsolatban áll távoli társával, a közvetlenül felette elhelyezkedő másik réteg számára szolgáltatásokat nyújt. Ezt a szolgáltató felületet ***interfésznek*** – másnéven Service Access Point, ***SAP*** – nevezzük. A referencia-modell meghatározza az egyes rétegek által végrehajtandó feladatokat, vagyis a rétegek célját, valamint a réteg – célra jellemző – nevét. (Az egyes rétegek szerepéről más fejezetekben szólunk.)

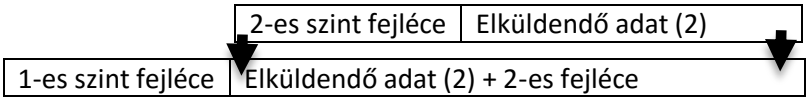
Szintek az OSI modellben

Megjegyzés:

1. ábra

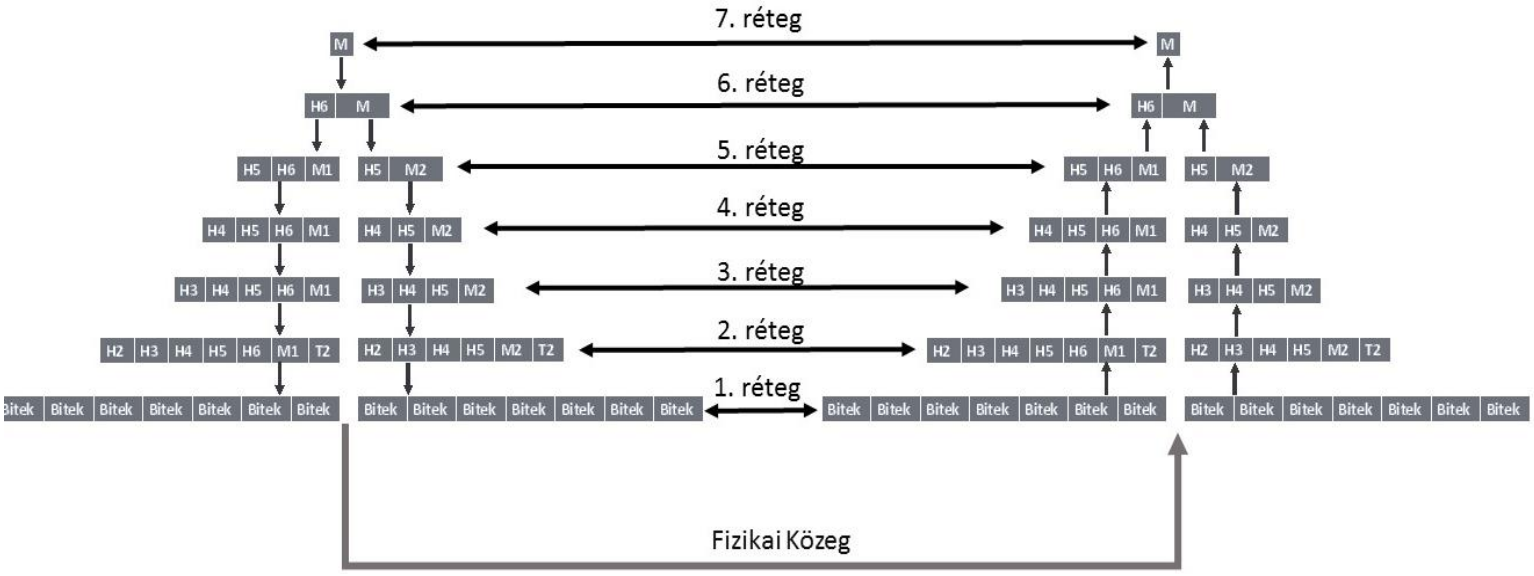
A modell egyes rétegei között van kommunikáció, így a 7-es réteg kapcsolatban van a 6-ossal a 6-os az 5-össel és a 4-essel és így tovább.

Az 1. ábra 2 eszköz közötti kommunikációját mutatja be, a rétegek közötti kommunikációt. Fontos tudnunk, hogy a rétegek csak a szomszédjukkal állnak közvetlen kapcsolatban, így csak a többi réteget felhasználva kommunikálhatnak más rétegekkel. Egy üzenet elküldése egy adott szintről a másik eszköz azonos szintjére csak az alattuk lévő csatornákkal lehetséges. Például, üzenet küldése a 2-es szintről:



Ezt aztán a fogadó állomás egyes szintje felbontja a fejlécben tárolt információk alapján dekódolja az adatot és továbbítja a 2-es szintnek. A 3-as ábra emutatja a teljes folyamatot, azaz a forrás Alkalmazási rétegétől indul kommunikáció, az adat beágyazódik a réteg fejlécével együtt a következő szint adat mezőjébe, majd ez ismétlődik egészen az 1-es szintig, ahol el lesz küldve a célállomásnak, aki kicsomagolja az adatokat és a fejlécinformációknak megfelelően továbbítja a megfelelő rétegeknek (LIFO elv).

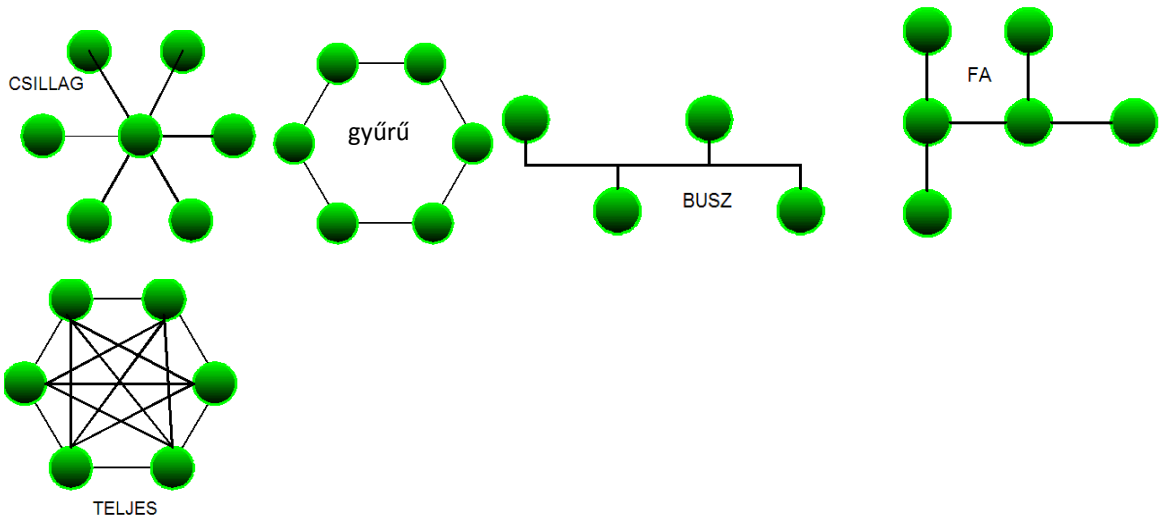
Hálózati Kommunikáció vázlata:



3. ábra

Hálózatok topológiája

A hálózatokat topológia szerint is feloszthatjuk. Főbb Hálózati topológiáink a csillag, gyűrű, fa, teljes elrendezésű hálózati topológia.



4. ábra

Az adatátvitel fizikai közegei:

- UTP/STP/koax rézvezeték
- üvegszál
- mikrohullám
- lézer
- szórt spektrumú sugárzás.

2. Fejezet:

OSI modell

Szintek az OSI modellben

5. ábra

Megjegyzés:

A rétegek hibaellenőrzést végeznek

OSI modell rétegei és feladatai

1. A fizikai réteg feladata a bitek kommunikációs csatornára való juttatása. Ez a réteg határoz meg minden, az eszközökkel kapcsolatos fizikai és elektromos specifikációt, beleértve az érintkezők kiosztását, a használatos feszültség szinteket és a kábel specifikációkat. A szinten "Hub"-ok, "repeater"-ek és "hálózati adapterek" számítanak a kezelt berendezések közé.
   A fizikai réteg által megvalósított fő funkciók:
   - felépíteni és lezárni egy csatlakozást egy kommunikációs médiummal.
   - részt venni egy folyamatban, amelyben a kommunikációs erőforrások több felhasználó közötti hatékony megosztása történik. Például, kapcsolat szétosztás és adatáramlás vezérlés.
   - moduláció, vagy a digitális adatok olyan átalakítása, konverziója, jelátalakítása, ami biztosítja, hogy a felhasználó adatait a megfelelő kommunikációs csatorna továbbítani tudja. A jeleket vagy fizikai kábelen – réz vagy optikai szál, például – vagy rádiós kapcsolaton keresztül kell továbbítani.

   Paralel SCSI buszok is használhatók ezen a szinten. A számos Ethernet szabvány is ehhez a réteghez tartozik; az Ethernetnek ezzel a réteggel és az adatkapcsolati réteggel is együtt kell működnie. Hasonlóan együtt kell tudni működnie a helyi hálózatokkal is.

2. Az adatkapcsolati réteg biztosítja azokat a funkciókat és eljárásokat, amelyek lehetővé teszik az adatok átvitelét két hálózati elem között. Jelzi, illetve lehetőség szerint korrigálja a fizikai szinten történt hibákat is. A használt egyszerű címzési séma fizikai szintű, azaz a használt címek fizikai címek (MAC címek) amelyeket a gyártó fixen állított be hálózati kártya szinten.
   Megjegyzés: A legismertebb példa itt is az Ethernet. Egyéb példák: ismert adatkapcsolati protokoll a HDLC és az ADCCP a pont-pont vagy csomag-kapcsolt hálózatoknál és az Aloha a helyi hálózatoknál.
   Az IEEE 802 szerinti helyi hálózatokon, és néhány nem-IEEE 802 hálózatnál, mint például az FDDI, ez a réteg használja a Media Access Control (MAC) réteget és az IEEE 802.2 Logical Link Control (LLC) réteget is. Ez az a réteg, ahol a bridge-ek és switch-ek működnek.
3. A hálózati réteg biztosítja a változó hosszúságú adat sorozatoknak a küldőtől a címzetthez való továbbításához szükséges funkciókat és eljárásokat, úgy, hogy az adatok továbbítása a szolgáltatási minőség függvényében akár egy vagy több hálózaton keresztül is történhet. A hálózati réteg biztosítja a hálózati útvonalválasztást, az adatáramlás ellenőrzést, az adatok szegmentálását/deszegmentálását, és főként a hiba ellenőrzési funkciókat. Az útvonalválasztók (router-ek) ezen a szinten működnek a hálózatban – adatküldés a bővített hálózaton keresztül, és az internet lehetőségeinek kihasználása (itt dolgoznak a 3. réteg (vagy IP) switch-ek). Itt már logikai címzési sémát használ a modell – az értékeket a hálózat karbantartója (hálózati mérnök) adja meg egy hierarchikus szervezésű címzési séma használatával. A legismertebb példa a 3. rétegen az Internet Protocol (IP).
   Egy adatcsomag eljutását A pontból B pontba, „hop”-okkal mérjük.
   1 hop = Az információ 2 direktkapcsolatú állomáson áthalad.
   A csomagokban álltalában van egy élettartam számláló, ami ha eléri a 64 hop-ot akkor a csomag valószínűleg eltévedt, ezért törlődik. Általában 10-nél kevesebb hop-pal el jut a világ bármely pontjára egy csomag.
4. A szállítási réteg biztosítja, hogy a felhasználók közötti adatátvitel transzparens legyen. A réteg biztosítja, és ellenőrzi egy adott kapcsolat megbízhatóságát. Néhány protokoll kapcsolat orientált. Ez azt jelenti, hogy a réteg nyomon követi az adatcsomagokat, és hiba esetén gondoskodik a csomag vagy csomagok újra küldéséről. A legismertebb 4. szintű protokoll a TCP.
5. A viszony réteg a végfelhasználói alkalmazások közötti dialógus menedzselésére alkalmas mechanizmust valósít meg. A megvalósított mechanizmus lehet duplex vagy félduplex, és megvalósítható ellenőrzési pontok kijelölési, késleltetések beállítási, befejezési, illetve újraindítási eljárások.
   (A mai OSI modellben a Viszonylati réteg a Szállítási rétegbe lett integrálva.)
6. A megjelenítési réteg biztosítja az alkalmazási réteg számára, hogy az adatok a végfelhasználó rendszerének megfelelő formában álljon rendelkezésre. MIME visszakódolás, adattömörítés, titkosítás, és egyszerűbb adatkezelések történnek ebben a rétegben. Példák: egy EBCDIC-kódolású szöveges fájl ASCII-kódú szövegfájllá konvertálása, vagy objektum és más adatstruktúra sorossá alakítása és XML formába alakítása vagy ebből a formából visszaalakítása valamilyen soros formába.

A mai OSI modellben az Adatmegjelenítési réteg az Alkalmazási rétegbe lett integrálva. (A mai OSI ezért valójában 5 rétegű mivel a régi 7 rétegű modell 5. rétege a 4. illetve a 6. rétege a 7. rétegbe integrálódott.)

Feladata:

- két számítógép között logikai kapcsolat létesítése
- párbeszéd szervezése
- vezérjelkezelés
- szinkronizálás

7. Az alkalmazási réteg szolgáltatásai támogatják a szoftver alkalmazások közötti kommunikációt, és az alsóbb szintű hálózati szolgáltatások képesek értelmezni alkalmazásoktól jövő igényeket, illetve, az alkalmazások képesek a hálózaton küldött adatok igényenkénti értelmezésére. Az alkalmazási réteg protokolljain keresztül az alkalmazások képesek egyeztetni formátumról, további eljárásról, biztonsági, szinkronizálási vagy egyéb hálózati igényekről. A legismertebb alkalmazási réteg szintű protokollok a HTTP, az SMTP, az FTP és a Telnet.

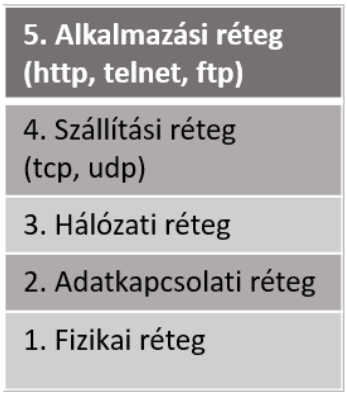
Le egyszerűsítve tehát:

1. **A fizikai réteg (**physical layer**)** A fizikai réteg a legalsó réteg, ezen zajlik a tényleges adatátvitel. Feladata a bitek hibamentes átvitele, azaz biztosítja, hogy az adó által küldött jeleket a vevő is azonosként értelmezze.
2. **Az adatkapcsolati réteg (**data link layer**)** Az adatkapcsolati réteg feladata az adatok kisebb egységekre, úgynevezett adatkeretekre (data frame) darabolása, és a keretek hibamentes célbajuttatása. Ezt úgy éri el, hogy a csomagokban adathalmazát egységnyi darabokra vágja, és majd minden kereten elvégez egy bonyolult matematikai műveletet, amelynek eredményét a keret végéhez illeszti. Ezt a számot CRC-nek (ciklikus redundancia control) nevezzük. A fogadó gép, miután megkapott egy keretet, ugyanazt a matematikai műveletet végzi el vele, mint a feladó gép. Saját eredményét összehasonlítja a keret végén található CRC-vel. Ha az elküldött, illetve a vevő oldalon számított eredmény megegyezik, akkor a vevő gép adatkapcsolati rétege egy úgynevezett nyugtakeretet küld a küldő gép adatkapcsolati rétegének, jelezve, hogy a keret hibamentesen megérkezett. Ha a küldő gép bizonyos időn belül nem kap nyugtakeretet, akkor az adatkeretet elveszettnek minősíti, és ismételten elküldi azt, forgalomszabályozást is végezve. A hibátlanul megérkező adatkereteket az adatkapcsolati réteg csomaggá illeszti össze, majd továbbítja azt a hálózati rétegnek.
3. **A hálózati réteg (network layer)** Vezérli a kommunikációs alhálózatok működését, legfontosabb feladata az útvonalválasztás a forrás és célállomás között. Ha az útvonalban eltérő hálózatok is vannak, akkor protokollátalakítást, -tördelést (fragmentation) is végez. Fontos megjegyezni, hogy míg az adatkapcsolati réteg az egymással kommunikáló távoli gépek között tartja a kapcsolatot és nem vesz tudomást az „útközben" elhelyezkedő gépekről, addig a hálózati réteg mindig csak egy szomszédos hosttal van kapcsolatban.
4. **A szállítási réteg (transport layer)** A végpontok közötti hibamentes adatátvitel biztosításáért felelős. A topológiát már nem ismeri, csak a két végpontban van rá szükség. Feladatai: összeköttetések felépítése, bontása, csomagok sorrendbe állítása, hibaérzékelés, helyreállítás és az adatáramlás vezérlése.
5. **A viszonyréteg (session layer)** Megteremti annak a lehetőségét, hogy két számítógép felhasználói kapcsolatot létesítsenek egymással, azaz a programok, pontosabban folyamatok összekapcsolását végzi el. Feladata az alkalmazások közti viszonyok felépítése, kezelése és lebontása.

6. **A megjelenítési réteg (presentation layer)** A fogadó rendszer számára biztosítja az adatok olvashatóságát. A megjelenítési réteg feladatai közé tartozik az adatok titkosítása, és visszafejtése is. A rétegek közül az egyetlen, amely megváltoztathatja az üzenet tartalmát.

7. **Az alkalmazási réteg (application layer)** Az alkalmazások számára biztosít hálózati szolgáltatásokat. Az adó oldalon elfogadja és feldolgozza a felhasználó által továbbítandó adatokat, a vevő oldalon pedig gondoskodik azok felhasználó felé történő továbbításáról. Pl.: fájlok gépek közötti másolása.

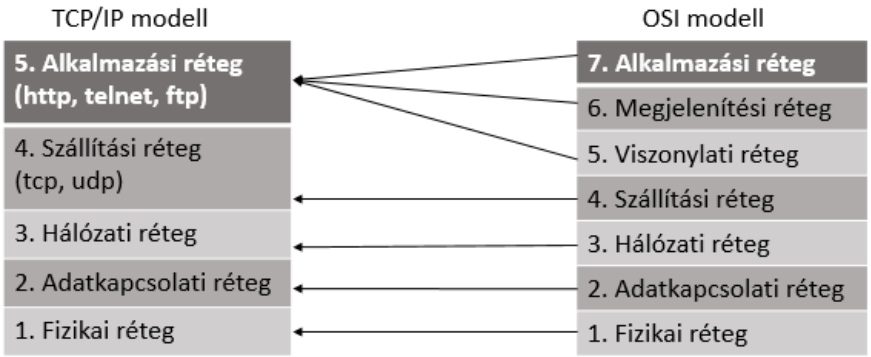
TCP/IP modell



1. A fizikai réteg továbbítja az adatkapcsolati rétegtől kapott kereteket a hálózaton. A fogadó oldalon ugyanez a folyamat játszódik le visszafelé, míg az adat a fogadó gép alkalmazásához nem ér. Eredetileg a fizikai és az adatkapcsolati réteg egy réteg volt, neve „Hoszt és hálózat közötti réteg”.
2. Az adatkapcsolati réteg szintén hozzárakja a kapott adathoz a saját fejlécét, és az adatot keretekre bontja. Ha a kapott adat túl nagy ahhoz, hogy egy keretbe kerüljön, feldarabolja és az utolsó keret végére egy úgynevezett tail-t kapcsol, hogy a fogadó oldalon vissza lehessen állítani az eredeti adatot.
3. A szállítási rétegtől kapott header-adat pároshoz hozzáteszi a saját fejlécét, amely arról tartalmaz információt, hogy az adatot melyik végpont kapja majd meg.(szokás az adatkapcsolati réteggel összevonva tekinteni, és Internetként nevezni, mely magába foglalja: (ip, icmp, arp, rarp)).
4. Az alkalmazási rétegtől kapott adat elejére egy úgynevezett fejlécet (angolul: header) csatol, mely jelzi, hogy melyik szállítási rétegbeli protokollal (leggyakrabban TCP vagy UDP) küldik az adatot.
5. Az alkalmazási réteg a felhasználó által indított program és a szállítási réteg között teremt kapcsolatot. Ha egy program hálózaton keresztül adatot szeretne küldeni, az alkalmazási réteg továbbküldi azt a szállítási rétegnek.

A TCP/IP protokoll:

Összehasonlítva az Osi modellel



6. ábra

Megjegyzés:

A rétegek hibaellenőrzést végeznek

Pár szó az RFC-ről:

Routing Information Protocol (RFC 1058)
A Routing Information Protocol (RIP) legfontosabb jellemzői:

- Távolságvektor alapú IGP protokoll.
- Régi, de folyamatosan fejlesztik, javítják.
- Metrika: érintett útválasztók (**hop**-ok) száma (azaz minden kapcsolat költsége 1).
- Max. 15 router hosszúságú optimális útvonalak esetén használható (16 = végtelen távolság).
- 30 másodpercenkénti **routing** információ küldés.
- A szomszédos útválasztó elérhetetlenségét hat hirdetési cikluson (180 sec.) keresztül történő "csendben maradása" jelzi.
- „**Triggered** update" a végtelenig számlálás idejének csökkentésére: Változás esetén nem várjuk ki a ciklusidőt, hanem azonnal továbbküldjük a változás információját (A változás "update"-et "**triggerel**"). (Speciális "**flag**"-ek és időzítési információk nyilvántartása szükséges)
- RIPv2 (RFC 1723): CIDR kompatibilis, a szomszédok közötti kommunikációra **authentikáció** előírható.

3. Fejezet

Az Ethernet története

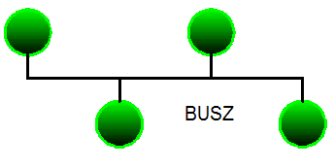
Az Ethernet az 1970-es évek elején kidolgozott ALOHANET-tel kapcsolatos fejlesztések eredménye. Bob Metcalfe és David Boggs 1976-ban tervezték meg és valósították meg az első helyi hálózatot a Xerox Palo Altó-i kutatási központjában. A nevét az éterről (angolul: ether), a 19. századi fizikusok által feltételezett, az elektromágneses sugárzások terjedésére szolgáló könnyű közegről kapta.

Az Ethernet esetén a közeg nem vákuum, hanem egy speciális koaxiális kábel volt, amely akár 2,5 km hosszú is lehetett (500 méterenként egy ismétlővel). A kábelre csavarozott adóvevőkkel legfeljebb 256 gépet lehetett csatlakoztatni. A központi kábelnek, amelyhez a gépek csatlakoztak sokcsatlakozós kábel (multidrop cable) volt a neve, és 2,94 Mb/s-os sebességgel tudott üzemelni. Ez az Ethernet olyan sikeres volt, hogy a DEC, az Intel és a Xerox 1978-ban megállapodott egy közös, 10 Mb/s-os Ethernet szabványban, amely a DIX szabvány nevet kapta. Ebből jött létre később az IEEE 802.3 szabvány 1983-ban.

Mivel a Xerox nem mutatott nagy érdeklődést az amúgy sikeres Ethernet iránt, Bob Metcalfe megalapította saját cégét, a 3Com-ot, hogy Ethernet csatolókat adjon el PC-khez. Eddig hozzávetőlegesen 100 millió darab kelt el belőlük.

Az Ethernet azóta is fejlődik, létezik már a 100 Mb/s-os és még nagyobb sebességű változat is, a kábelezés is változott.

Busz topológia elvén működik:



7. ábra

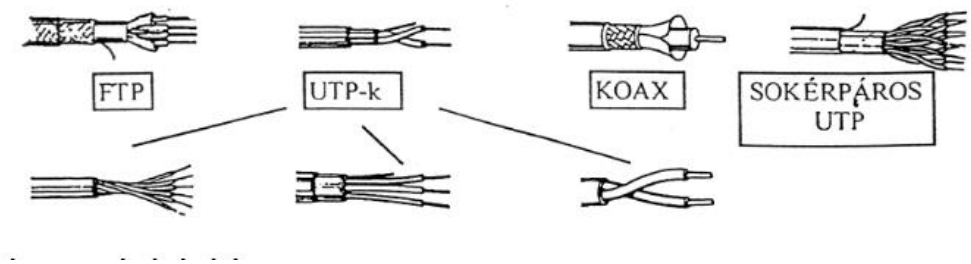
A számítógépek kommunikációja általában úgy történik, hogy az egyik gép a hálózat összes gépének kiküld egy üzenetet, melyet csak az a gép olvas el, amelynek a címe szerepel az üzenet fejlécében, a többi gép eldobja az üzenetet és nem reagál. Fontos, hogy a kommunikáció a hálózaton váltakozva oda-vissza történik, elkerülve az ütközést, azaz hogy két üzenet kioltsa egymást vagy megzavarják. Erre különböző hibaellenőrző megoldások vannak. Az üzenetek elején, a fejlécben szerepel mindig a célállomás címe, így ha ez nem beazonosítható, mert egy másik üzenet megzavarja, elrontja, akkor az üzenetre nem érkezik válasz ezért a küldő gép megismétli az üzenetet. Ekkor addig történik az üzenetküldés, amíg egy sorrend be nem áll és sikeres üzenetküldés eredményeként válasz érkezik.

Adatátviteli közegek

- Vezetékes átviteli közegek:
  + Csavart érpár (utp)
  + Koaxiális kábel
  + Üvegszálas kábel
- Vezeték nélküli adatátviteli közegek:
  + Infravörös
  + Lézer átvitel
  + Rádióhullám
  + Szórt spektrumú sugárzás
  + Műholdas átvitel
  + Bluetooth

Vezetékes Adatátviteli közegek

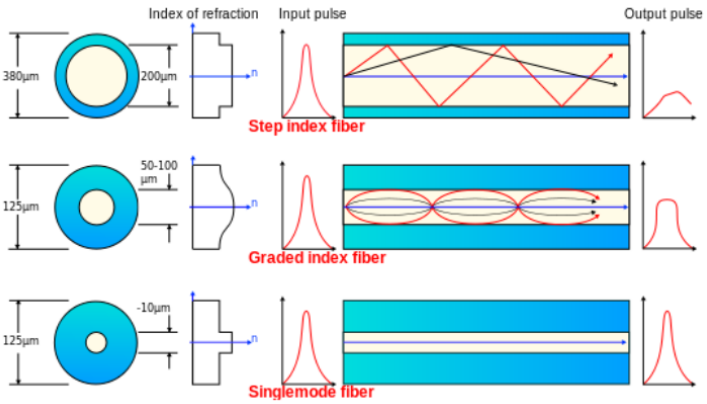
1. Csavart érpár (utp)
   - Két szigetelt sodort érpárból áll, a sodrástól függően kategóriákba sodorható:
     1. Kategória: Hang minőségű átvitel (pl.: telefonvonalak)
     2. Kategória: 4 Mbit/s -es sebességűátvitelre képes vonal (Helyi beszélgetés)
     3. **Kategória**: 10 Mbit/s -es sebességűátvitelre képes vonal (Ethernet)
     4. Kategória: 20 Mbit/s -es sebességűátvitelre képes vonal (16Mbit/s Token ring)
     5. Kategória: 100 Mbit/s -es sebességűátvitelre képes vonal (Fast Ethernet)
   - A kábelek álltalában 4 csavart érpárt tartalmaznak, a zavarokat árnyékolással csökkenthetjük.
   - ilyenek pl.:



8. ábra

2. Koaxiális kábel
   - 2 típusa van:
     1. Alapsávú koaxiális kábel (Digitális jelátvitelre alkalmas)
     2. Szélessávú koaxiális kábel (Analóg jelátvitelre alkalmas pl.: televíziós adás)
3. Üvegszálas kábel
   - Működési elve: fényimpulzusok segítségével egy adó (fényforrás) és egy vevő (fényérzékelő) között információ továbbítható. A fényjeleket a vevő értelmezi és értelmes jellé alakítja vissza.

9. ábra



Működése:

A fény beesési szöge és az impulzus együttese képzi az értelmezendő jelet. a kibocsátott jel az üvegszálas vezeték belsejében a falakról visszaverődik mindig ugyanabba a beesési szögben.

Vezetéknélküli adatátviteli közegek

1. Infravörös adatátvitel
   - IrDA: Ipari szabvány vezetéknélküli adatátvitelhez, infravörös fény segítségével.
   - 30°-os nyílásszögig kb. 1 méteren belül működik, nem kell tartani külső zavaró jeltől.
2. Lézeres adatátvitel
   - Nagy adatátviteli képességük és magas biztonságuk miatt szokták alkalmazni.
3. Rádióhullámú adatátvitel
   - Mikrohullámokkal dolgozunk, 2-40 GHz.
   - Adótornyok segítségével szórjuk, egy adótorony kb. 100km-t fed le.
   - A jelterjedés befolyásolható a jel frekvenciájával és a környezeti zajokkal, ezekhez modulálni kell a jelet, hogy minimális legyen az interferencia és jó minőségű legyen a sugárzott jel. A jelek erősíthetők átjátszó antennák segítségével.
4. Szórt spektrumú sugárzás
   - Kisebb távolságokra alkalmazandó (1 km)
   - Széles frekvenciasávot, és helyi hálózati megoldást alkalmaz
5. Műholdas adatátvitel
   - Egyenlítő felett 36000km magasan keringő műholdak biztosítják, a keringési sebességük azonos a föld forgási sebességével.
   - Tipikus sávszélesség: 500MHz
   - hátránya hogy lehallgatható, és a jel csak nagy késéssel ér oda és vissza a távolság miatt.
6. Bluetooth
   - Rövid hatótávolságú adatcseréhez használt nyílt vezetéknélküli szabvány, mely számítógépek, mobilok közötti kommunikáció biztonságos nem lehallgatható formája.

CSMA/CD protokoll

Vivőjel-érzékelős, többszörös hozzáférésű, ütközés-érzékelő protokoll. Az állomás figyeli a hálózatot. Ha nincs rajta csomag, akkor kezdeményezi az adást. Ha két állomás egyszerre próbálkozik, akkor mindkét állomás egy ideig vár, majd újra megpróbálja elküldeni a csomagot. Nagy hálózat esetén ez a módszer lelassítja a működést.

Azaz a CSMA/CD egy közös kommunikációs csatorna több fél közötti elosztásának módját és szabályait meghatározó rendszer.
A CSMA/CD szabályrendszere biztosítja, hogy a közös csatornán kommunikáló felek képesek legyenek a szimultán adások és az ezekből következő ütközések detektálására, valamint ezek elhárítására. Amennyiben a CSMA/CD csatornán több fél azonos idejű adási kísérlete miatt ütközés jön létre, úgy az érintett felek külön-külön, de véletlenszerűen meghatározott ideig felfüggesztik adási kísérleteiket abban a reményben, hogy így a következő próbálkozás alkalmával már nem egyszerre próbálnak majd meg adni. Amennyiben az újabb adási kísérlet során ismét ütközés jön létre, úgy az adni kívánó felek a korábbiaknál egyre tovább várnak ki - így csökkentve az újabb ütközések esélyét.

Ethernet keretek és címek felépítése

Az ethernet hálózatokon az adatok keretekben jutnak el a címzettől a feladóig. A keretek valójában nem mások, mint mezőkre osztott bitsorozatok. A mezőkben a következők szerepelhetnek:

1. Az előtag (preamble) váltakozva tartalmaz egyeseket és nullákat. 7 darab 10101010 tartalmú bájtból álló sorozat. A 10 Mbit/s-os és kisebb sebességű Ethernet-megvalósításoknál az órajel szinkronizálása ennek a mezőnek a segítségével történik. Az Ethernet gyorsabb változatai szinkronműködésűek, ezeknél időzítési információkra nincs szükség; ennek ellenére, a kompatibilitás érdekében a mező megmaradt.

2. Az előtagot egy egyoktetből álló mező a keretkezdő (start frame delimiter) követi, amely az időzítési információk végét, a keret tényleges kezdetét jelzi. Tartalma az 10101011 bitsorozat.

3. Ezután a cél (destination) és küldő (source) állomás 48-bites címei következnek. Az Ethernet hálózaton minden állomást egy egyedi, 48-bites (6 bájtos) ún. MAC (**Media Access Control**) cím azonosít. Ezen címek kiosztását az IEEE kontrollálja.

4. A hossz/típus mezőt kétféle célra lehet használni. Ha értéke a decimális 1536-nál, vagyis a hexadecimális 0×600-nál kisebb, akkor a benne szereplő érték hosszt ad meg, egyébként típus értékként azt adja meg, hogy az Ethernet folyamatainak lezárulása után melyik felsőbb rétegbeli protokoll fogja kapni az adatokat. A hossz a mezőt követő adatrészben található bájtok számát adja meg.

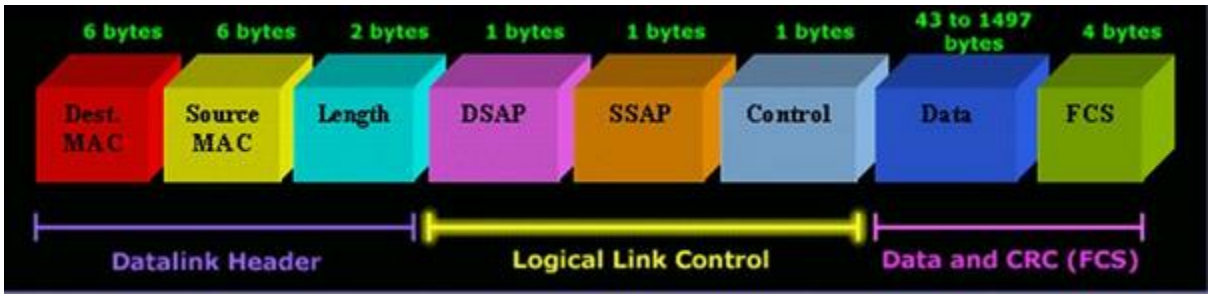
5. Az adat mező és a szükség szerinti kitöltés hossza tetszőleges lehet, azonban a keret mérete nem haladhatja meg a felső mérethatárt. A maximális átviteli egység (maximum transmission unit, MTU) az Ethernet esetében 1500 oktett, az adatok mérete tehát ezt nem haladhatja meg. A mező tartalma nincs meghatározva. Ha nincs elég felhasználói adat ahhoz, hogy a keret mérete elérje a minimális kerethosszt, akkor előre meg nem határozott mennyiségű adat kerül beillesztésre, közvetlenül a felhasználói adatok mögé. Ezt a többletadatot nevezzük kitöltésnek. Az Ethernet keretek hosszának 64 és 1518 oktett között kell lennie.

6. A keret végén szereplő FCS (Frame Check Sequence - Keret Ellenőrző Sorozat) mezőben egy 4 bájton CRC ellenőrző összeg helyezkedik el. Ha a vevő által számolt és a keretben lévő összeg nem egyezik, a keret eldobásra kerül.

Ethernet címek felépítése:

10. ábra

Ethernet csomag felépítése:



11. ábra

Ethernet hálózatok kiegészítő elemei:

- ***Hub***: A hub (középpont, csomópont) a számítógépes hálózatok egy hardvereleme, amely fizikailag összefogja a hálózati kapcsolatokat. Másképpen szólva a hub a hálózati szegmensek egy csoportját egy hálózati szegmensbe vonja össze, egyetlen ütközési tartományként láttatja a hálózat számára. Leegyszerűsítve: az egyik csatlakozóján érkező adatokat továbbítja az összes többi csatlakozója felé. Ez passzívan megy végbe, anélkül, hogy ténylegesen változtatna a rajta áthaladó adatforgalmon.
- ***Bridge***: A hálózati híd (angolul network bridge) hálózati szegmenseket tud összekötni az OSI modell Adatkapcsolati réteg szintjén. Ethernet hálózatokban pontosan azokat az eszközöket nevezzük hídnak, amelyek megfelelnek IEEE 802.1D szabványnak. Különbség a hub és bridge között: Bár mindkét eszköz számítógép-hálózatokat kapcsol össze, más módon teszik azt. A hálózati híd az OSI modell második, tehát az Adatkapcsolati rétegében operál, míg a router az OSI modell 3. más szóval a hálózati rétegében tevékenykedik. Ez azt jelenti, hogy a híd a hardveres MAC-cím alapján irányítja a kereteket, a router pedig a szoftveresen hozzárendelt IP-címek alapján. Ennek egyik következménye, hogy a hidak nem tudnak különbséget tenni alhálózatok között, a routerek viszont igen. Számítógép-hálózatok tervezésekor dönthetünk úgy is, hogy az egyes szegmenseket hidakkal kapcsoljuk össze, s ezáltal egy nagy hálózatot hozunk létre, viszont a szegmenseket routerekkel is összeköthetjük, s így azok külön-külön alhálózatok lesznek.

- Ha egy gépet át kell helyezni egyik szegmensből a másikba, akkor a routeres megoldás esetén új IP-címet kell hozzárendelni, viszont a hidas megoldásnál nem kell semmit újrakonfigurálni.
- ***Switch***: Az adatátviteli kapcsoló vagy switch (ejtsd: szvics) egy aktív számítógépes hálózati eszköz, amely a rá csatlakoztatott eszközök között adatáramlást valósít meg. Többnyire az OSI-modell adatkapcsolati rétegében (2. réteg, esetleg magasabb rétegekben) dolgozik. Magyar jelentése: vált, kapcsol.
  A fizikai rétegbeli feladatokat ellátó hubokkal szemben az Ethernet switchek adatkapcsolati rétegben megvalósított funkciókra is támaszkodnak. A MAC címek vizsgálatával képesek közvetlenül a célnak megfelelő portra továbbítani az adott keretet; tekinthetők gyors működésű, többportos hálózati hídnak is. Portok között tehát nem fordul elő ütközés (mindegyikük külön ütközési tartományt alkot), ebből adódóan azok saját sávszélességgel gazdálkodhatnak, nem kell megosztaniuk azt a többiekkel. A broadcast és multicast kereteket természetesen a switchek is floodolják az összes többi portjukra.
  Egy switch képes full-duplex működésre is, míg egy hub csak half-duplex kapcsolatokat tud kezelni. Különbség még, hogy a switchek egy ASIC (Application-Specific Integrated Circuit) nevű hardver elem segítségével jelentős sebességeket érhetnek el, míg a HUB nem más mint jelmásoló, ismétlő. A fontos funkciók közé tartozik még a hálózati hurkok elkerülésének megoldása (lásd STP), illetve a VLAN-ok kezelése.
  Alapvető feladata:

    1. csomagokban található MAC címek megállapítása
    2. MAC címek és portok összerendelése (kapcsoló-tábla felépítése)
    3. a kapcsoló-tábla alapján a címzésnek megfelelő port-port összekapcsolása
    4. adatok ütközésének elkerülése, adatok ideiglenes tárolása

Virtuális LAN hálózatok (VLAN)

A virtuális helyi hálózatok az OSI modell 2. szintjén találhatók, azonban a rendszergazdák gyakran konfigurálják úgy, hogy a **VLAN** és az IP-hálózat között kölcsönös kapcsolat legyen, így azt a benyomást keltheti mintha a **VLAN**-nak köze lenne a 3. réteghez. A **trönk (trunk)** olyan hálózati kapcsolat, amely több, csomagszinten azonosított **VLAN** fogalmát viszi át. Ezek általában a nem felhasználó által kezelt eszközök között mennek, például: switch-switch, router-router, vagy switch-router.
A VLAN-okat a VTP(Virtual Trunking Protokoll) kezeli, biztosítja a VLAN konfiguráció egységességét. Speciális úgyevezett **Trunkkereteket** használ, a **vlanok** hozzáadásának, törlésének és átnevezésének kezelésére. A VTP biztosítja a nemkívánatos VLAN egyezések elkerülését ezzel megakadályozva az esetleges illetéktelen vagy véletlen behatolást a hálózatba.

Előnyei:

- Konzisztens VLAN-konfiguráció az egész hálózatban
- VLAN-trönkölés különböző típusú átviteli közegeken keresztül
- A VLAN-ok pontos nyomon követése, monitorozása
- Dinamikus jelentés a hálózathoz hozzáadott VLAN-okról
- Újonnan hozzáadott VLAN-ok **Plug-and-play** konfigurációja

A VLAN létrehozása előtt szükséges egy VTP tartomány létrehozása, mely folytonosan trönkölt (trunk) switch-ekből áll melynek tartományneve megegyezik. A tartományban lévő switch-ek közösen menedzselik a VLAN-al kapcsolatos információkat.

Részletesebben:
https://hu.wikipedia.org/wiki/Virtuális_helyi_hálózat#Cisco_VLAN_Trunking_Protocol_.28VTP.29

4.Fejezet

Internetcímzés

Interneten való kommunikációra a számítógépeknek szüksége van valamilyen megkülönböztetésre, erre a MAC cím bonyolult lenne és rendszertelen. Ezért a számítógépeket az egyes hálózati eszközök különböző „IP” címekkel látja el. Ezek a címek csoportokba oszthatóak. Az otthoni kis hálózatok „C” osztályú hálózatokat alkalmaznak aza egy kép ip-címe általában 192.168.x.x azaz az utolsó két hely hozzáférhető a hálózatok számára, így meghatározhatják hogy egy hálózatban a gépünk hol helyezkedik el. pl 192.168.0-es hálózatban 100-as címmel ellátott gép ip-címe: 192.168.1.100. Ezt többnyire a számítógép automatikusan lekéri a hálózati eszköztől: éppen szabad ip-címet kapja 100-254-ig. Egy hálózat 255 címmel rendelkezhet.Ez alapján egy lokális hálózati eszköz legfeljebb 255 eszközt tud kiszolgálni egy hálózaton belül. amennyiben ez kevés, lehetőség van új hálózatot létrehozni 2-es hálózattal melynek címe: 192.168.2.0. itt újra 255 címet tudunk kiosztani.

Osztályok:

5 különböző osztályt különböztetünk meg

- A: Az 1.0.0.0 és 127.0.0.0 közötti hálózatokat foglalja magában. Itt az első szám a hálózat száma. Az „A” osztályban nem osztják ki a következő IP címeket Internetes hálózat céljára:

  10.0.0.0 – belső hálózatokban lehet használni (Intranet);

  127.0.0.0 – belső hálózati tesztelési címek (loopback).

  Az „A” osztályban így 125 darab hálózatot lehet létrehozni, melyekben egyenként 2^{32}-2, azaz 16,777,214 darab IP címet lehet kiosztani. Nem osztható ki gépeknek a x.0.0.0 és a x.255.255.255 IP cím. Az első a hálózat címe, a második az úgynevezett „broadcast” cím. Ha erre a címre van egy üzenet címezve, akkor azt a hálózatban lévő összes gép megkapja.

  Az „A” osztályba tartozó hálózatok olyan nagyok lehetnek, hogy csak néhány ilyen hálózat létezik a világon (pl. IBM hálózata).

- B: A 128.0.0.0 és a 191.255.0.0 közötti hálózatokat foglalja magában. Itt az első két szám a hálózat száma. A „B” osztályban nem osztják ki a következő IP címeket Internet-es hálózat céljára:

  172.16.0.0 – 172.31.0.0 – belső hálózatokban lehet használni (Intranet).

  Az „B” osztályban így 16384-16, azaz 16368 darab hálózatot lehet létrehozni, melyekben egyenként 2^{16}-2, azaz 65,534 darab IP címet lehet kiosztani. Nem osztható ki gépeknek a x.y.0.0 és a x.y.255.255 IP cím az „A” osztályhoz hasonlóan.

- C: A 192.0.0.0 és a 223.255.255.0 közötti hálózatokat foglalja magában. Itt az első három szám a hálózat száma. A „C” osztályban nem osztják ki a következő IP címeket Internet-es hálózat céljára:

  192.168.1.0 – 192.168.255.0 – belső hálózatokban lehet használni (Intranet).

  Az „C” osztályban így 2,097,152-255, azaz 2,096,897 darab hálózatot lehet létrehozni, melyekben egyenként 2^8-2, azaz 254 darab IP címet lehet kiosztani. Nem osztható ki gépeknek a x.y.z.0 és a x.y.z.255 IP cím az „A” osztályhoz hasonlóan.

- D: A 224.0.0.0 - 239.0.0.0 közötti címek tartoznak hozzájuk, ***multicasting*** eljárás céljaira vannak fenntartva.
- E: A 240.0.0.0 - 255.0.0.0 közötti címek tartoznak hozzájuk, melyek az Internet saját céljaira fenntartott címek.

A „D” és „E” osztályokban nem oszthatók ki IP címek.

Ahhoz, hogy IP címről el lehessen dönteni, hogy „A”, „B”, vagy „C” osztályba tartozik-e, hálózati maszkot használnak. A hálózati maszk és az IP cím közötti logikai AND művelet a hálózat címét adja vissza. Így az „A” osztályú hálózat maszkja általában 255.0.0.0, a „B” osztályú hálózaté 255.255.0.0, míg a „C” osztályú hálózaté 255.255.255.0

A hálózati maszk (pl. Intranetben) arra is használható, hogy korlátozzuk a kioszthatő IP címek számát. Például: Egy 30 darab gépet tartalmazó hálózat esetén választhatjuk hálózati maszknak a 255.255.255.224-et. Ebben az esetben az utolsó szám binárisan 1110 0000 – azaz 128+64+32=224 ahol az utolsó 5 biten 0-tól 31-ig lehet számokat kiosztani. Mivel a 0 nem osztható ki, így 30 gépet lehet megcímezni.

Alhálózatok

Az alhálózatok kisebb azonos hosszúságú részekre bontott címtartomány. Az ip cím host részének legmagasabb helyiértékű bitjéből néhányat az alhálózat azonosítására használunk. az új hálózati csomópont határvonalának pozícióját a hálózati maszk megadásával jelöljük.

Példa az alhálózatokra:

Hálózat ip-címe: 192.42.11.0/24

192	42	11	0
11000000	00101010	00001011	00000000

Alapértelmezett hálózati maszk: 255.255.255.0

255	255	255	0
11111111	11111111	11111111	00000000

Használjunk 3 bitet hálózat azonosításra:

255	255	255	224
11111111	11111111	11111111	11100000

Ezáltal összesen 8 alhálózat kialakítására van lehetőség:

bitek	1	1	1	0	0	0	0	0
alhálózatok száma	2	4	8	16	32	64	128	256

Alhálózatok címei (8 részre bontva):

1.	197.45.11.0	197.45.11.1-30
2.	197.45.11.32	197.45.11.33-62
3.	197.45.11.64	197.45.11.65-94
4.	197.45.11.96	197.45.11.97-126
5.	197.45.11.128	197.45.11.129-158
6.	197.45.11.160	197.45.11.161-190
7.	197.45.11.192	197.45.11.193-222
8.	197.45.11.224	197.45.11.225-254

DHCP

A dinamikus állomáskonfiguráló protokoll (angolul Dynamic Host Configuration Protocol) egy számítógépes hálózati kommunikációs protokoll. Az IETF RFC 1541, majd később a RFC 2131 határozza meg. Ez a protokoll azt oldja meg, hogy a TCP/IP hálózatra csatlakozó hálózati végpontok (például számítógépek) automatikusan megkapják a hálózat használatához szükséges beállításokat. Ilyen szokott lenni például az IP-cím, hálózati maszk, alapértelmezett átjáró stb. A DHCP szerver-kliens alapú protokoll, nagy vonalakban a kliensek által küldött DHCP kérésekből, és a szerver által adott DHCP válaszokból áll. A DHCP-vel dinamikusan oszthatóak ki IP-címek, tehát a hálózatról lecsatlakozó számítógépek IP-címeit megkapják a hálózatra felcsatlakozó számítógépek, ezért hatékonyabban használhatóak ki a szűkebb címtartományok.

3 féle IP-kiosztás lehetséges DHCP-vel:

- kézi (MAC cím alapján)
- automatikus (DHCP-vel kiadható IP-tartomány megadásával)
- dinamikus (IP-tartomány megadásával, de az IP címek „újrahasznosításával”) [Wikipedia]

Működése:

A DHCP kliens-szerver elven dolgozik. Mikor egy állomás (kliens) csatlakozik a hálózathoz, és DHCP-vel konfigurálja azt, kiküld egy DHCP kérést broadcast üzenetben, melyre csak a dhcp-t kezelő hálózati eszközök (Szerver, Router) válaszolnak. Amennyiben van DHCP szerver a hálózatban, akkor az válaszol, a válasz pedig tartalmazza:

- A kiosztott IP-cím és alhálózati maszk
- Alapértelmezett átjáró
- Érvényességi idő
- DNS szerver IP-címe

Az állomás (kliens) küld egy visszaigazolást a szervernek, hogy elfogadja-e a kiosztott IP-címet, ezután a szerver eltárolja az adatokat és a kiosztás sikerességét, és jelzi a kliensnek, hogy a kiosztott IP-címet használatba veheti.

ARP, RARP

Az ARP a routerek egy olyan táblája melyben eltároljuk az adott rajta áthaladó kommunikáció adatait, a végpontok Fizikai címét (Mac cím) és a hozzá tartozó elérhető útvonalat, így ha újabb kérés érkezik a routerhez, melyben az adott Mac című számítógépet keresik, akkor az ARP tába segítségével már tudni fogja melyik porton keresztül kell tovább küldeni az üzenetet, nem szükséges broadcast küldése.

Az RARP protokoll az ARP protokoll fordítottja, a táblázatban az van eltárolva, hogy melyik ip címhez melyik fizikai cím tartozik.

Részletesebben: https://hu.wikipedia.org/wiki/ARP; https://hu.wikipedia.org/wiki/RARP

Fizikai és internet címek összerendelése

A hálózati eszközöknek, mint például hálózati kártya, a gyártó rögzít egy fizikai címet mely egyedi és az adott eszközhöz tartozik, számokbólés betűkből álló cím. Ezt a címet bonyolult lenne használni egy összetett hálózat esetén ezért a hálózati eszközök, a folytonosság céljából ip címet használnak, ezzel leegyszerűsítve a hálózati kommunikációt. Amikor egy hálózati eszköz kapcsolódik az internethez, kap egy ip címet, és csak a forgalomirányítók, vagy speciális hálózati eszközök (dns szerver) tárolják el a számítógép mac címét (Fizikai cím).

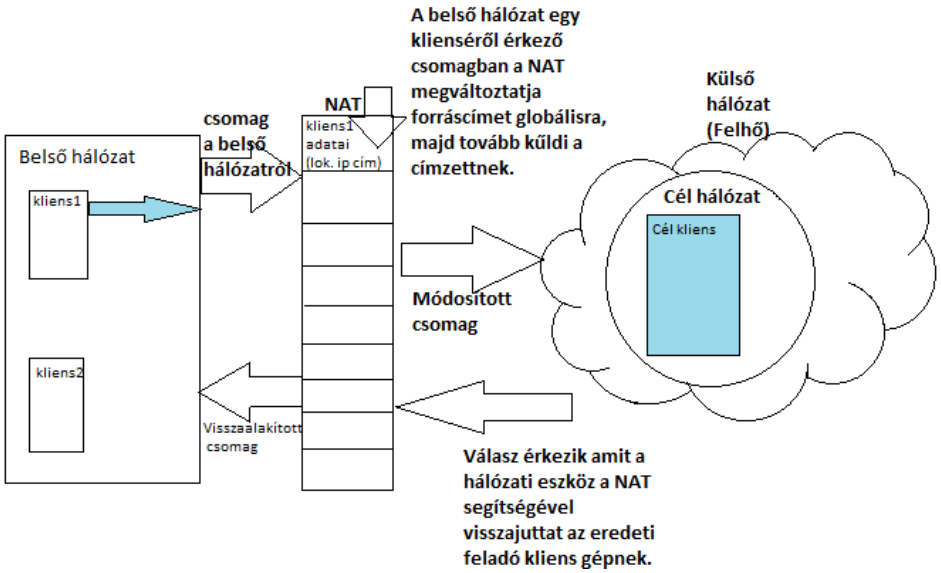
Az interneten IP cím segítségével történik a kommunikáció, Amikor egy számítógép keres egy adott fizikai című hálózati eszközt vagy számítógépet, a router elkezdi keresni, egy táblázatban, melyen el vannak tárolva a címekhez tartozó elérhetőségek, ahol megtalálja, ott ip cím segítségével üzenetet küld, az üzenetben tárolva a saját mac címét melyet a céleszközhöz tartozó hálózati kiszolgáló eltárol a saját listájában ezzel megkönnyítve a későbbi kommunikációt is.

Részletesebben: https://hu.wikipedia.org/wiki/MAC-c%C3%ADm

NAT

A hálózati címfordítás (angolul **Network Address Translation**, röviden **NAT**) a csomagszűrő tűzfalak, illetve a címfordításra képes hálózati eszközök (pl. router) kiegészítő szolgáltatása, mely lehetővé teszi a belső hálózatra kötött gépek közvetlen kommunikációját tetszőleges protokollokon keresztül külső gépekkel anélkül, hogy azoknak saját nyilvános IP-címmel kellene rendelkezniük. Címfordításra akár egyetlen számítógép is képes, így valósítható meg például az internet-kapcsolat megosztás is, amikor a megosztó gép a saját publikus címébe fordítja bele a megosztást kihasználó kliens gép forgalmát. Kialakulásának oka, hogy az IPv4 tartománya viszonylag kevés, $2^{32} = 256^4$ azaz 4.294.967.296 db egyedi IP címből áll, beleértve az összes broadcast cím és a külső hálózatra nem route-olható belső címtartományokat is, azaz az interneten globálisan használható címek összessége így még kevesebb. A NAT úgy működik, hogy a hálózati címfordító a belső gépekről érkező csomagokat az internetre továbbítás előtt úgy módosítja, hogy azok feladójaként saját magát tünteti fel, így az azokra érkező válaszcsomagok is hozzá kerülnek majd továbbításra, amiket a belső hálózaton elhelyezkedő eredeti feladónak ad át. Minden esetben szükség van egy aktív hálózati eszközre, amely folyamatosan figyeli az érkező csomagokat, és azok feladói és címzettjei alapján elvégzi a módosításokat a csomagon. A belső hálózatnak olyan címtartományt kell adni, amelyet minden hálózati eszköz a nemzetközi szabványoknak megfelelően belsőnek ismer el, és így azokat nem irányítja közvetlenül a külső hálózat felé. A címfordítás segítségével megoldható, hogy akár egy egész cég teljes belső hálózati forgalma egyetlen külső IP cím mögött legyen, azaz gyakorlatilag egyetlen külső címet használ el egy több száz gépes hálózat. A belső forgalomban természetesen szükség van az egyedi belső címekre, de erről csak a címfordítást végző hálózati eszközöknek kell tudnia, kifelé ennek részletei már nem látható információk. Így létrejöhet olyan gazdaságos konfiguráció is, hogy egy viszonylag nagy cég teljes külső címfoglalása 10-20 db cím, míg a belső forgalmukban akár több ezer belső cím is lehet. Nagy előnye ennek a technikának, hogy ugyanazt a belső tartományt nyugodtan használhatja bárki más is, amíg mindegyik egyedi külső cím mögé van fordítva, ez nem okoz zavart. Akár az összes NAT-ot használó cég belső hálózatában lehet minden gép a 10.0.0.0 tartományban, ha kifelé valóban egyedi címmel látszanak.

A NAT vázlatos működése

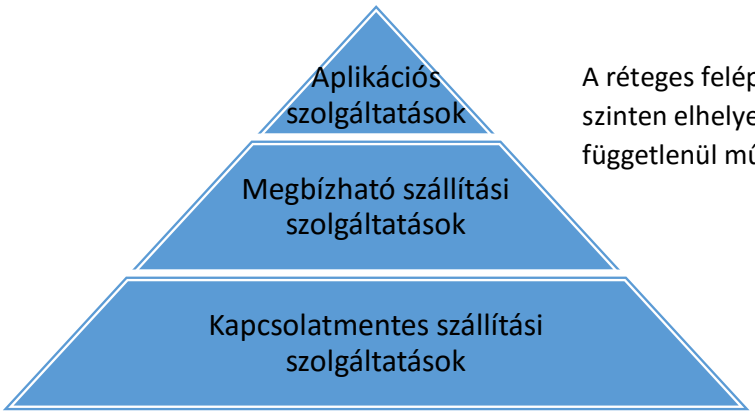


5. Fejezet

Kapcsolatmentes adatátvitel

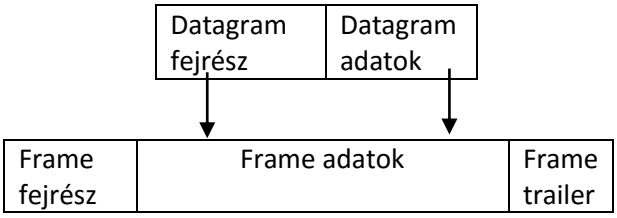
Ez az egyik legalapvetőbb Internet szolgáltatás, melyet az IP (Internet Protokoll) nyújt. Csomagok (datagramok) szállítását végzi, „felelősség nélkül” azaz nem figyeli, hogy az általa elküldött csomag célba is ér e, nem feladata a valós kapcsolat ellenőrzése. A kapcsolatmentesség azt jelenti, hogy az egyes adatcsomagokat függetlenül juttatja célba. A csomagvesztés tényéről semelyik felet nem értesíti.

Elhelyezkedése:



A réteges felépítés lehetővé teszi, hogy a különböző szinten elhelyezkedő protokollok egymástól függetlenül működhessenek

A csomagok az átvitel során a fizikai szállítóközeg csomagjában (Ethernet frame) utaznak, ezt nevezik csomagolásnak (Encapsulation).



Az ip datagram felépítése:

Verzió	Fejléc hossza	Folytonosság(0-7)	D	T	R	--	Teljes hossz	
Azonosítás							Flag-ek (DF,MF)	Datagramdarabeltolás
Élettartam		Protokoll					Fejrész ellenőrző összege	
Feladó IP címe								
Címzett IP címe								
Opcionális mezők (IP)								
IP fejrész								

D-T-R: átvitel jellege

D: Alacsony késleltetésű átvitel (0/1)

T: Nagy sávszélesség (0/1)

R: Nagy megbízhatóság (0/1)

Datagram tördelése és visszaállítása

Egy datagram maximális mérete 64000 oktet, azonban a fizikai szállítási közeg általában csak 1500 oktetet támogat. Ha a datagram mérete meghaladja a fizikai szállítási közeg maximális csomagméretét (MTU: Maximum Transfer Unit) akkor tördelni kell a datagramot. Az így kapott darabokat **Fragment**-eknek nevezik. A széttördelt datagram szerkezetleg megegyezik az eredetivel, csak néhány mező tatalma változik.

Azonosító: Azonosítja a datagramot, azonban a tördelt darabokban megegyeznek a mező értékei.

Datagramdarabeltolás: A darab helyét (offset) adja meg az eredeti datagramban.

Flagek: Első bitje a „Do not fragment” (DF) bit. Ha 1 akkor nem tördelhető. Utolsó bitje a „More Fragments” (MF), ami az utolsó fragment kívételével mindig 1-es, azt jelzi hogy van e több fragment utána.

Élettartam (TTL): Minden router 1-el csökkenti, ha 0 akkor eldobja, és értesíti a feladót.

Protokoll: Szállítási adat típusa.

IP opciók

Az opciók szolgálnak olyan ritka, IP szintű funkciók megvalósítására, melyeknek nem volt érdemes a minden csomagban jelen lévő fejlécben helyen fenntartani. Az opciókat minden állomás köteles megérteni és feldolgozni, nem implementációjuk, csupán jelenlétük választható. Az RFC 791 a következő opciókat definiálja:

1. Security. A csomag hitelesítéséhez szükséges információk.
2. Source routing. A feladó által megadott útvonalon, állomások megadott listáján halad végig a csomag. Két vállfaja van, a szigorú (strict) és a laza (loose). Az első esetben csak a listán felsorolt állomásokon haladhat végig a csomag és ha két szomszédosnak felsorolt állomás nem szomszédos, akkor a csomag elvész és egy „Source routing failed" ICMP csomag küldődik a feladóhoz. A második esetben ha a listán két szomszédosnak feltüntetett állomás a valóságban nem szomszédos, akkor is továbbítódik a csomag a lista következő eleméhez, de a router-ek által kijelölt útvonalon.
3. Útvonalrögzítés. A csomag által érintett állomások IP címe rögzül a csomagban.
4. Időbélyeg
5. Stream ID. Egy 16 bites azonosító, főként más, folyam(kapcsolat)orientált hálózatokkal való együttműködés segítése miatt.

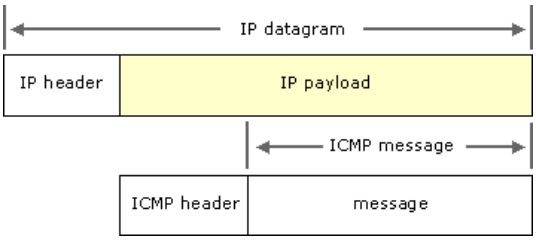
Részletesebben: icourse.cuc.edu.cn/networkprogramming/lectures/Unit4_IP.pdf

Az ICMP protokoll

Az ICMP egy interneten használt protokoll, melynek segítségével értesülhetünk a hibákról illetve azok típusáról, valamint hálózati diagnosztizálásban lehet a segítségünkre. Az ICMP (az UDP-hez hasonlóan) datagram-orientált kommunikációs protokoll, mert egyáltalán nem garantált a csomagok megérkezése vagy sorrendje. Az ICMP (a TCP-hez és az UDP-hez hasonlóan) az IP-t használja borítékként (ICMP csomagok csak IP hálózaton mehetnek). Az ICMP-t részletesen az RFC 792-ben definiálták. (Wikipedia)

Az ICMP-üzenetek küldése a következő helyzetekben automatikusan megtörténik:

- Egy IP-datagram nem ér célba.
- Egy IP-útválasztó (átjáró) nem képes a datagramokat továbbítani a jelenlegi átviteli sebességen.
- Egy IP-útválasztó átirányítja a küldő állomást, hogy az a célhoz egy kedvezőbb utat találjon.



12. ábra

Fontosabb ICMP üzenetek:

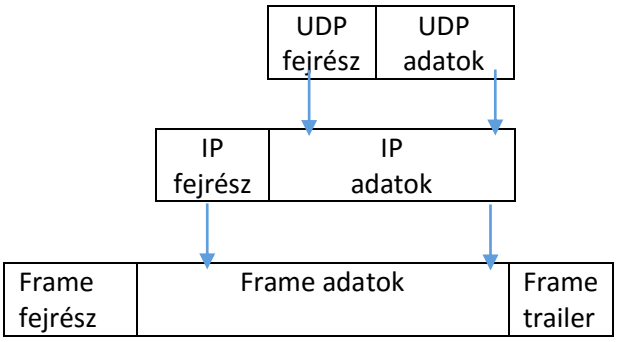
- Viszhangkeresés: Eldönti hogy egy IP-csomópont elérhető e a hálózaton
- Viszhangválasz: Válasz a viszhangkeresésre
- A cél nem elérhető: Tájékoztatja a feladót, hogy nem kézbesíthető a datagram
- Forráselnyomás: Túlterheltség esetén az értesíti a küldő állomást, hogy csökkentse a datagramok küldésének sebességét.
- Átirányítás: Értesíti a feladót egy előnyösebb útvonalról.
- Időtúllépés: Jelzi hogy egy datagram élettartama (TTL) lejárt.

Felhasználás: http://softwareonline.animare.hu/cikk.aspx?id=2979

6. Fejezet

UDP

A User Datagram Protocol (UDP) az internet egyik alapprotokollja. Feladata datagram alapú szolgáltatás biztosítása, azaz rövid, gyors üzenetek küldése. Jellemzően akkor használják, amikor a gyorsaság fontosabb a megbízhatóságnál, mert az UDP nem garantálja a csomag megérkezését. Ilyen szolgáltatások például a DNS, a valós idejű multimédia átvitelek, vagy a hálózati játékok.



Az UDP olyan szegmenseket (segment) használ az átvitelhez, amelyek egy 8 bájtos fejrészből, valamint a felhasználói adatokból állnak. A két port a végpontok forrás- és a célgépen belüli azonosítására szolgál. Amikor egy UDP-szegmens megérkezik, akkor az adatmezejét a szállítási entitás kézbesíti a címzett portra kapcsolódó folyamatnak. A folyamatok a bind vagy más hasonló primitív használatával kapcsolódhatnak rá egy portra. A portokra úgy gondoljunk, mint olyan postaládákra, amelyeket az alkalmazások ki tudnak bérelni csomagok fogadására. Az UDP használatának tulajdonképpen az a legnagyobb előnye a IP használatával szemben, hogy a fejrészben megtalálható a feladó és a címzett portszáma is.

0	16	31
Feladó port	Címzett port	
UDP hossza	UDP ellenőrzőösszege	
Felhasználói datagram adatok		

Feladó címe: Feladó utasítás hoszton belüli portcíme, ha nem adjuk meg, akkor értékét 0-ra kell állítani.

Címzett címe: Címzett utasítás hoszton bebelüli portcíme.

UDP hossza: UDP datagram hossza oktetben mérve. (fejrész + adat)

UDP ellenőrzőösszeg: Esetleges hibák felismerése a fejlécben

A portszámokat tartalmazó mezők nélkül a szállítási réteg nem tudná, hogy mit kezdjen a csomaggal. A segítségükkel azonban a beágyazott szegmenseket a megfelelő alkalmazásoknak tudja kézbesíteni. A forrás portjára elsősorban akkor van szükség, amikor választ kel küldeni a feladónak. A választ küldő folyamat úgy tudja megadni, hogy az üzenete a célgép melyik folyamatának szól, hogy a bejövő szegmens forrásport mezőjét átmásolja a kimenő szegmens célport mezőjébe. Az UDP-szegmens hossza mező a 8 bájtos fejrész és az adatmező együttes hosszát tartalmazza. A legrövidebb lehetséges hossz 8 bájt, ami csak a fejrészt tartalmazza. A legnagyobb lehetséges hossz 65 515 bájt, ugyanis az IP-csomagok mérethatára nem teszi lehetővé a legnagyobb, 16 biten ábrázolható számú bájt használatát. Az UDP ellenőrző összeg mező használata nem kötelező, de használata növeli a megbízhatóságot. Az ellenőrző összeget a fejrészre, az adatra és egy IP-álfejrészre számítják ki. Számítása során az UDP ellenőrző összeg mezőt 0-nak veszik, és az adatmezőt további 0 bájttal egészítik ki, ha az UDP-szegmens hossza mező páratlan értékű. Az ellenőrző összeg algoritmus a szegmenst 16 bites szavakra bontja, majd egyszerűen kiszámolja az egyes komplemens összegüket és végül a végeredmény egyes komplemensét veszi. Ennek következményeképpen, amikor a vevő ezt a számítást a teljes szegmensre végrehajtja, beleértve az ellenőrző összeg mezőt, az eredmény 0 lesz. Az ellenőrző összeg mező 0-t tartalmaz abban az esetben, ha nem számították ki, köszönhetően annak a szerencsés egybeesésnek, hogy a 0-nak kiszámolt összeg egyes komplemense csupa 1-esből áll. IPv6 esetén némileg különböző, de ezzel hasonló módon járunk el. Az álfejrész beszámítása az UDP ellenőrző összegbe segít megtalálni a tévesen kézbesített csomagokat, azonban megsérti a protokollhierarchiát, hiszen az IP-címek az IP-réteghez tartoznak, nem pedig az UDP-réteghez. A TCP is pontosan ezt az álfejrészt használja az ellenőrző összeg számításához. A TCP a hibamentes átvitelkor pozitív nyugtázás újraküldését hajtja végre. Az adó elküld egy csomagot, ugyanekkor elindít egy időzítést is, majd nyugtára vár. Ha a nyugta nem érkezik meg az időzítő lejárta előtt, feltételezi, hogy a csomag elveszett, és újból elküldi a csomagot (elkerülve a csomagok többszöröződését, sorszámmal látjuk el őket). A hálózat jobb kihasználása érdekében több csomagot is elindít a feladó, nem várja meg a nyugtákat, ezt csúszóablakos (sliding window) megoldással hajtja végre. Az egyidőben úton lévő csomagok számát az ablak mérete határozza meg.

Socket

A számítógép-hálózatokban a szoftvercsatorna, csatlakozó, Internet socket vagy egyszerűen socket egy Internet Protocol-alapú számítógépes hálózatban, például az interneten valamely kétirányú folyamatközi kommunikációs (IPC) hálózati folyam végpontja. Az operációs rendszer minden socketet egy kommunikáló alkalmazásprocesszhez vagy -szálhoz rendel. A socket címe egy IP-cím (a számítógép helye) és egy portszám (ami az alkalmazáshoz köthető) együttese, hasonlóan ahhoz, ahogy a telefonkapcsolat egyik felét meghatározhatja egy telefonszám és a választott mellékállomás kombinációja. A socket lényegében egy absztrakció, amit a Berkeley BSD Unix vezetett be. Ennek segítségével egy alkalmazói program egyszerűen hozzáférhet a TCP/IP protokollokhoz.

A socket lényegében egy IP-címből, ami egy gazdagépet, és egy úgynevezett portcímből áll, ami egy alkalmazást azonosít az adott gazdagépen. Ezzel a két adattal az Interneten egyértelműen azonosíthatóak az alkalmazások.

Típusai:

- Datagram socketek, vagy kapcsolat nélküli socketek, ezek User Datagram Protocolt (UDP) használnak.
- Stream socketek, vagy kapcsolat-orientált socketek, ezek Transmission Control Protocolt (TCP) vagy Stream Control Transmission Protocolt (SCTP) használnak.
- Raw socketek (vagy Raw IP socketek), jellemzően útválasztókban és más hálózati eszközökben találhatók. Itt a szállítási réteget kihagyják, és a csomagok fejlécét nem vágják le, hozzáférhető marad az alkalmazásnak. Alkalmazásszintű példák az Internet Control Message Protocol (a ping műveletről ismert ICMP), az Internet Group Management Protocol (IGMP) és az Open Shortest Path First

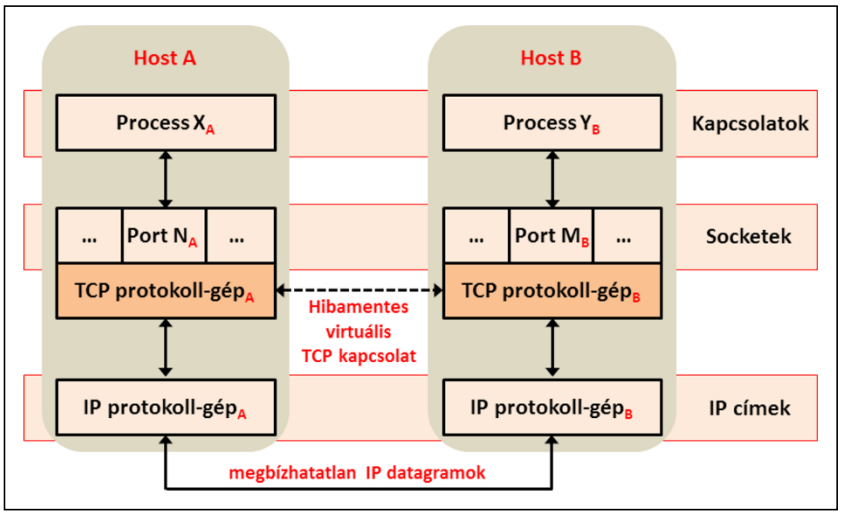
Egyéb részletek és információk: https://hu.wikipedia.org/wiki/Socket

TCP

A TCP célja, felhasználása

A Transmission Control Protocol, röviden TCP a szállítási réteg protokollja. (A TCP/IP protokoll-család legidősebb tagja. Az IP – és más segédprotokollok – ténylegesen a TCP támasztotta igények kiszolgálására később született.) Felhasználója számára megbízható vég-végpontok közti adatfolyam szolgáltatást nyújt a megbízhatatlan IP szolgálat felhasználásával. Olyan alkalmazások használják, amelyeknek az UDP – IP-re épített – megbízhatatlan szállítási szolgáltatása nem megfelelő. A TCP kapcsolat orientált, vagyis a két végpont között egy kétirányú adatkapcsolatot – virtuális áramkört – hoz létre. A TCP képes a végpontok igényeihez igazodva szabályozni az átvitt adatok mennyiségét, ezt folyamat-befolyásolásnak – flow control – nevezzük. A TCP képes továbbá a hálózatban fellépő torlódások adatátvitelre gyakorolt negatív hatását a lehetőségek között optimálisan közömbösíteni.

A TCP kapcsolat koncepcionális felépítése és a kapcsolat szerepe



Megjegyzés:

A TCP kapcsolat két végpontja egy-egy host, azon belül egy-egy process.

A kapcsolatot a hostok IP címe, és a gépen belül alkalmazott socket – TCP port – címek együttesen azonosítják.

A TCP nyújtotta szolgáltatás lényegi jellemzői

A TCP lényegi működési jellemzői öt pontban foglalhatók össze:

1. **Adatfolyam orientált (Stream oriented)**

   Az alkalmazás által továbbítani kívánt adatokat bitek – helyesen oktetek – sorozatának tekintjük. Az adatfolyam-szolgáltatás pontosan ugyanazt a bitsorozatot adja át a célgépen futó alkalmazásnak, amelyet az adatokat küldő gép adott át az átviteli szolgáltatásnak.

2. **Virtuális áramköri kapcsolat (Virtual Circuit Connection)**

   Telefon kapcsolathoz hasonlítható. Az alkalmazások az operációs rendszerhez fordulnak, kérik az átviteli szolgáltatást. Az operációs rendszer kérésére a végponti protokoll-gépek kapcsolatba lépnek egymással, ellenőrzik, mindkét fél kész-e a kapcsolat létrehozására, majd megállapodnak a részletekben. Ezután a protokoll-gépek értesítik a végpontokon futó két alkalmazást, hogy a kapcsolat létrejött, megkezdhetik a kétirányú adatátvitelt.

   A virtuális áramkör megléte alatt a két protokoll-gép együttműködése biztosítja, hogy az átvitt adatok hibátlanok legyenek. Csak a helyrehozhatatlan hibákat jelentik az alkalmazásoknak (ekkor azonban a virtuális áramkör megszakad).

   Azért nevezzük a kapcsolatot virtuális áramkörnek, mert az alkalmazások úgy látják, mintha az egy dedikált, külső körülmények által meg nem zavart kapcsolat lenne. A megbízhatóságot az adatfolyam átviteli protokoll biztosítja.

3. **Pufferelt átvitel (Buffered Transfer)**

   Az alkalmazások (közel) tetszőleges mennyiségű adatot adhatnak át átvitelre a szállítási szolgáltatásnak. Az adatokat a protokoll-gép pufferben gyűjti, majd a hatékonyságot szem előtt tartva kisebb-nagyobb csomagokban továbbítja azokat. Szükség esetén akár egy-egy oktet is átvitelre kerül (pl. egy billentyűleütés). Az erre szolgáló úgynevezett push mechanizmus kényszeríti a protokollt, hogy a puffer megtelte előtt vigye át az adatot. Ilyenkor a vételi oldalon működő protokoll-gép késleltetés nélkül átadja az adatot az alkalmazásnak.

4. **Strukturálatlan adatfolyam (Unstructured Stream)**

   A TCP által kézbesített adat nem strukturált. A szállítási szolgáltatás semmit sem tud az átviendő adatok tartalmáról, azok struktúrájáról. Az alkalmazásoknak kell megegyezniük az adatok szerkezetében, és megérteniük az adatfolyamot, felismerni az abban lévő esetleges adat-határokat.

5. **Egyidejű kétirányú kapcsolat (Full Duplex Connection)**

   A TCP kapcsolat egyidejű adatfolyam-átvitelt biztosít mindkét irányban. Az alkalmazások lezárhatják az egyik irányú adatfolyamot, ha kívánják (az ettől kezdve fél-duplex lesz). A kétirányú kapcsolat azért is előnyös, mert az ellenkező irányban haladó adatfolyam vezérlő – szolgálati – üzenetet is továbbíthat. Ez – a piggy-backing-nek nevezett – megoldás csökkenti a hálózati forgalmat.

Portok, kapcsolatok és végpontok

A TCP lehetővé teszi, hogy több alkalmazás egyidejűleg kommunikáljon, mivel a beérkező üzenetet a megfelelő process-hez továbbítja. Ennek érdekében a TCP – az UDP-hez hasonlóan – az adott gépben futó process hálózati címzésére bevezetett port címeket használja, ez alapján azonosítja az üzenet végső címzettjét. (A TCP és UDP portok – 0…65535 értéket tartalmazó – csoportja külön halmazt alkot, vagyis egy adott TCP port-cím nem azonos az ugyanazon értékű UDP port-címmel!).

A TCP port azonban önmagában nem azonosítja a cél objektumot. A TCP a kapcsolat (connection) fogalmát használja fel az azonosításhoz. Ezeket a kapcsolatokat két végponttal azonosítjuk. A végpontot pedig két összetartozó – címet jelölő – egész szám azonosítja. A számpár jelentése: (*host*, *port*), ahol

- host = az adott végpontot képviselő gép (a kapcsolatot tartó interface!) IP címe,
- port = a kapcsolatban felhasznált TCP port az adott gépben.
  Pl.: (**192.191.73.37, 25**)

Magát a virtuális kapcsolatot a két végpont együttes címe – vagyis 4 adat – azonosítja.

Pl.: **(192.191.73.37, 25)** és **(192.190.99.55, 1071)**

Egy lehetséges másik kapcsolat ugyanazon a két végponti gépen:

Pl.: **(192.191.73.37, 25)** és **(192.190.99.55, 12645)**

Amint látható, ahhoz, hogy a TCP a virtuális áramkörök között különbséget tegyen, elegendő, ha a kapcsolatot azonosító 4 szám közül az egyik különbözik!

A hálózat felhasználói programokból történő elérését egyszerűsítendő, az operációs rendszer a file műveletekhez hasonló értelmű – open, close, read, write, seek, stb. – eljárásokat nyújt a felhasználói programok írói számára. E csatolófelületet *socket interface* néven említik, a kapcsolatot azonosító címeket *socket címnek* nevezik.

Az UDP-hez hasonlóan a TCP is használja a fenntartott – más néven: jól ismert – port-számokat (*well-known port assignments*).

A TCP passzív és aktív felhasználási módjai

Egy szerveren futó, szolgáltatást nyújtó alkalmazás indulásakor egy *passzív megnyitást – passive socket open* – hajt végre, jelezve, hogy kész bejövő kapcsolatot fogadni. Ekkor az operációs rendszer egy TCP port-számot rendel a majdani kapcsolat e végpontjához. Ezután a szolgáltatást megvalósító process – a virtuális kapcsolat tényleges létrejöttéig – lényegében tétlenül várakozik.

A szolgáltatást felhasználni szándékozó alkalmazás a másik végponton az ottani operációs rendszert egy *aktív megnyitásra – active socket open* – utasítja, hogy létrejöjjön a virtuális kapcsolat. Erre a két végponton működő TCP protokoll-gép felépíti a kapcsolatot. Ezután megkezdődhet az alkalmazások adatcseréje. A kapcsolatot két címpár együttese azonosítja, ezek a következők: ($\mathbf{IP_{szerver}, Port_{szerver}}$) és ($\mathbf{IP_{kliens}, Port_{kliens}}$), ahol:

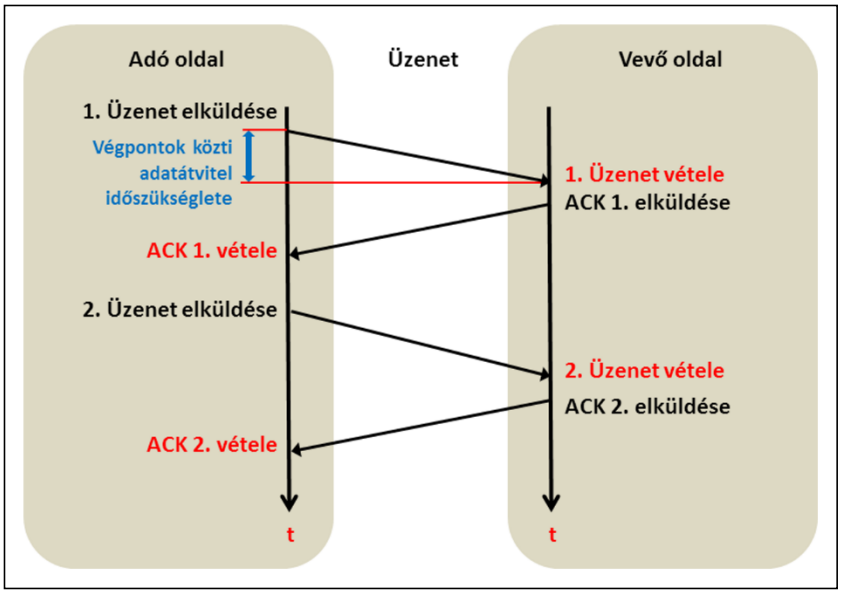
1. $\mathbf{IP_{szerver}}$ = a szolgáltatást nyújtó gép IP címe,
2. $\mathbf{Port_{szerver}}$ = a szolgáltatás – többnyire jól ismert – well-known – port címe,
3. $\mathbf{IP_{kliens}}$ = a szolgáltatást igénybevevő másik végponti gép IP címe,
4. $\mathbf{Port_{kliens}}$ = a szolgáltatást igénylő process – operációs rendszer adta – port címe.

A hibamentes adatátvitel biztosítása

Hogyan tud a TCP protokoll-gép megbízható szállítási szolgáltatást nyújtani az általa felhasznált IP megbízhatatlan csomag átviteli szolgáltatásával?

A megoldás: **pozitív (kumulatív) nyugtázás, ismételt (visszalépéses) átvitellel [positive (cumulative) acknowledgement with (step-back) retransmission]**. A működés elvét az alábbi ábrák szemléltetik:

Pozitív nyugtázás újraküldéssel

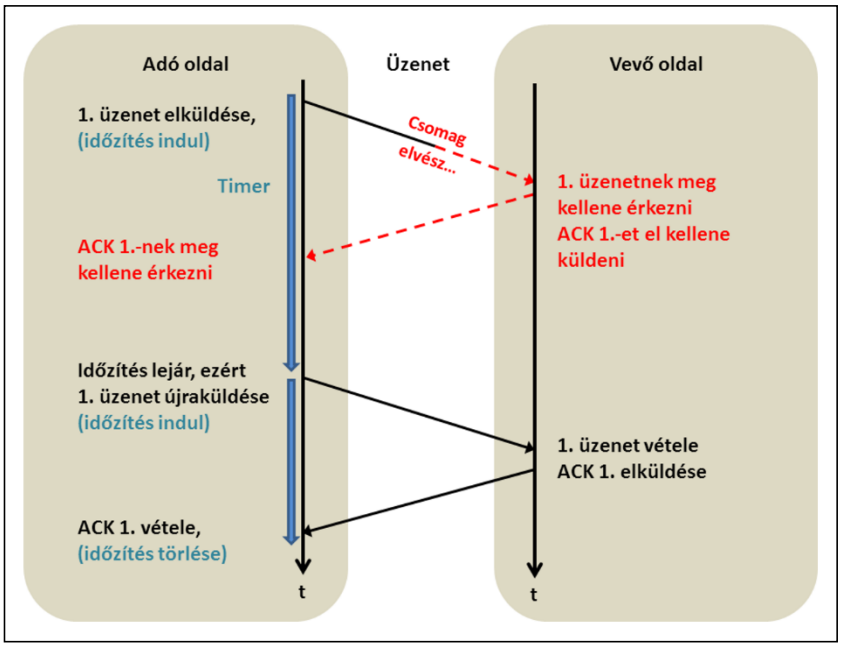


Megjegyzés:

A két függőleges időtengely a végpontokban eltelő idő múlását jelképezi. A csomag kiküldését szemléltető nyilak ferde volta az adatátvitel időszükségletére utal.

A vételi oldalon működő protokoll-gép nyugtát (Acknowledgement, röviden ACK) küld a feladónak, ha adat érkezik. Az adó minden átküldött üzenetet nyilvántart, és vár a nyugtára.

Újraküldés, ha a csomag elvész



Megjegyzés:

Ha az időzítéshez viszonyítva túl nagy a hálózati késleltetés, az üzenetek – adat és nyugtája egyaránt – a Timer lejárta miatt hibátlan átvitel esetén is megkettőződhetnek, a ténylegesen felesleges újraküldés miatt.

A nyugták nem a hibát, hanem az adatok hibátlan vételét jelzik. Ezt pozitív nyugtázásnak nevezzük.

Minden egyes csomag elküldésekor a TCP egy újabb időzítőt indít el, majd ha ennek lejártáig nem érkezik meg a csomag vételét nyugtázó üzenet, újraküldi az adatot.

Az adóként működő protokoll-gép minden üzenetet sorszámmal lát el, a vevőnek pedig emlékeznie kell, mely sorszámú üzenetek érkeztek már meg. A nyugtában a vevő szerepét játszó protokoll-gép visszaküldi a kapott adat sorszámát az adónak, így az a nyugtákat és az elküldött üzeneteket egymáshoz rendelheti.

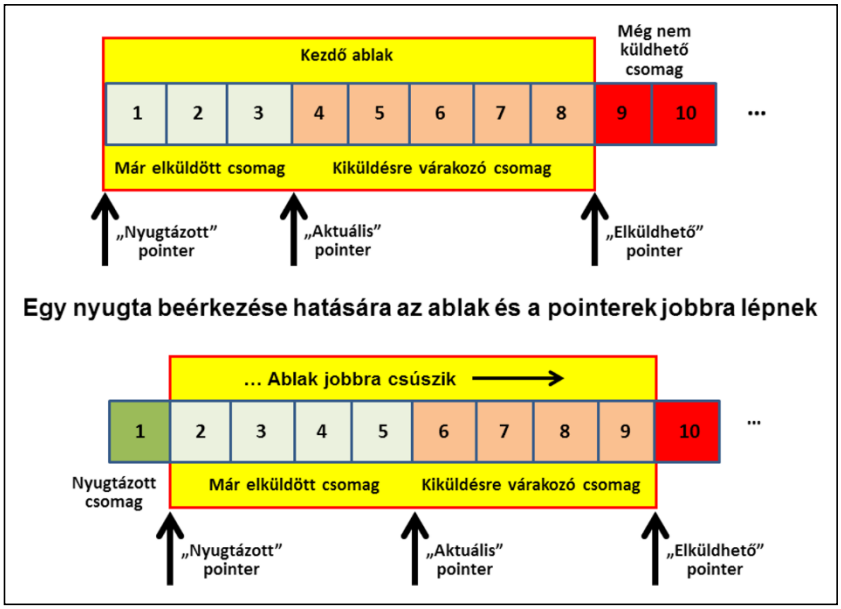
Csúszó ablak (Sliding Window)

Minden átküldött üzenet után várakozni a nyugtára, mielőtt a következő üzenetet elküldenénk, nem hatékony. Így egyszerre csak egyetlen üzenet lehetne úton, az adónak tétlenül kellene várakoznia a

nyugta megérkezéséig. A csúszó ablakos technikával több üzenetet is elküldhetünk, mielőtt a nyugtára várnánk. Így az adó folyamatosan dolgozhat.

A protokoll-gép az átküldendő üzenetek sorozatára egy – a virtuális kapcsolaton átvitt csomagok számához viszonyítva kisméretű – ablakot fektet, és az ablakba eső összes adatcsomagot elküldi.

Csúszó ablakos protokoll működése példaként 8 csomag méretű ablakkal



Megjegyzés:

Az ábrán látható pointerek követik az adatátvitel folyamatát, magyarázatuk a szövegben olvasható.

A két szélső pointer közti távolság az ablak méretét jelöli ki, a példában ez az érték 8.

Az adó működését három pointer segíti, ezeket „*Nyugtázott*”, „*Aktuális*” és „*Elküldhető*” pointernek nevezzük. Értelmezésük a következő:

- A Nyugtázott pointertől balra álló adatok átvitele a vevőhöz már sikeresen lezajlott, erről a beérkező nyugta tanúskodott. Mivel ezek az adatok már a vevő puffereiben vannak, megőrzésük a vevő felelőssége, az adó oldalon tárolni azokat már nem szükséges.
- A Nyugtázott és az Aktuális pointer között található adatcsomagok kiküldése már megtörtént, de az átvitel sikeréről tudósító nyugták még nem érkeztek meg az adóhoz.
- Az Aktuális pointer arra az adatcsomagra mutat, amelynek hálózatra küldése éppen most zajlik.
- Az Aktuális és Elküldhető pointerek között található adatcsomagok azok, amelyek az ablak adta határok között még a vonalra küldhetők anélkül, hogy újabb nyugta érkezne.
- Az Elküldhető pointertől jobbra található adatcsomagok kívül esnek az ablak határain, ezért adásuk nem kezdődhet meg, még akkor sem, ha az adó egyébként szabad lenne. Ezek a csomagok majd akkor küldhetők el, ha – egy nyugta beérkezésének hatására – az ablak jobbra mozdul.

Egy nyugta beérkezése a szélső pointerek elmozdulását eredményezi. A baloldali – Nyugtázott – pointer annyival lép jobbra, hogy új pozíciója a nyugtában érkezett csomag-sorszám és az azt követő sorszámú csomag határát jelezze. (Vegyük észre: a beérkező nyugta egyidejűleg több csomag nyugtázását is jelentheti. Ekkor a nyugtában álló számérték az ez előtti nyugtában álló sorszámhoz képest nagyobb, mint 1.) Egy adott sorszámot jelző nyugta beérkezése egyidejűleg valamennyi ennél kisebb sorszámú csomag nyugtázását is jelenti. Erre utal az összegző – vagy kumulatív – nyugtázás elnevezés.

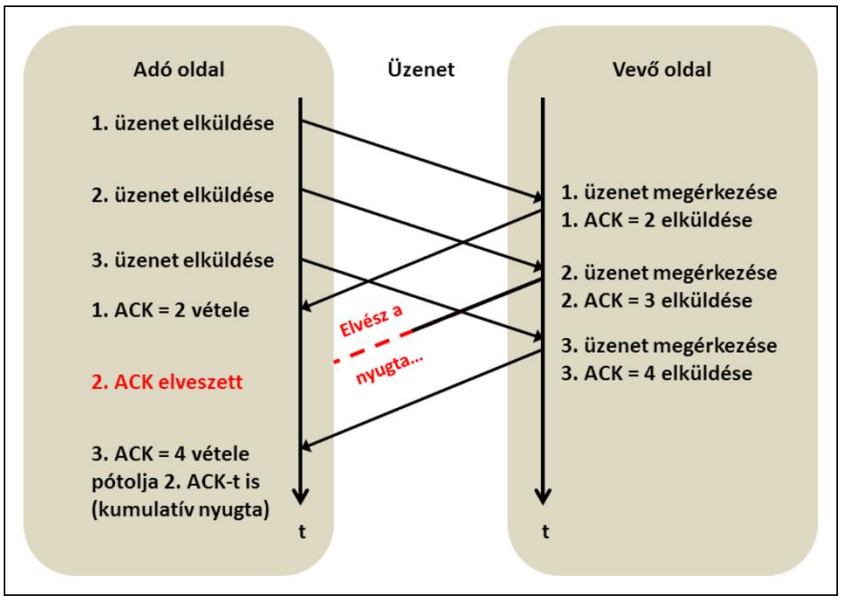
A baloldali pointer elmozdulását követi a jobboldali – Elküldhető – pointer is úgy, hogy e két pointer közti távolság megegyezzék az ablakmérettel.

A leírt elmozdulások hatására az ablak balszélén álló csomag(ok) nyugtázottá válnak, míg az ablak jobbszélén kívül álló csomag(ok) belépnek az ablakba, vagyis elküldhetővé válnak.

A leírtakat úgy is értelmezhetjük, hogy az ablak az átviendő adatcsomagok felett jobbra csúszik. Erre utal a csúszó ablak elnevezés. Amint látható, az ablak alatt található csomagok két állapot egyikében lehetnek: már kiküldöttek, illetve kiküldésre várakozók.

A középen található Aktuális pointer – a szélsők mozgásától függetlenül – halad balról jobbra, ahogy az adó a vonalra küldi az ablakban található csomagokat. Helyzete azonban szigorúan korlátozott: nem léphet ki a két szélső pointer által határolt intervallumból. Ha ez mégis megesne, az a protokoll-gépek hibás működését – szinkronvesztését – jelenti, ami a virtuális áramkör elbontásához vezet.

3 csomag elküldése csúszó ablakos protokoll felhasználásával



Megjegyzés:

Az ábrán látható esetben elvész a 2. ACK = 3 üzenet, de hiányát pótolja a később érkező 3. ACK = 4, amely egyszerre nyugtázza a 2. és 3. üzenetet. Ezt összegző – kumulatív – nyugtázásnak nevezzük.

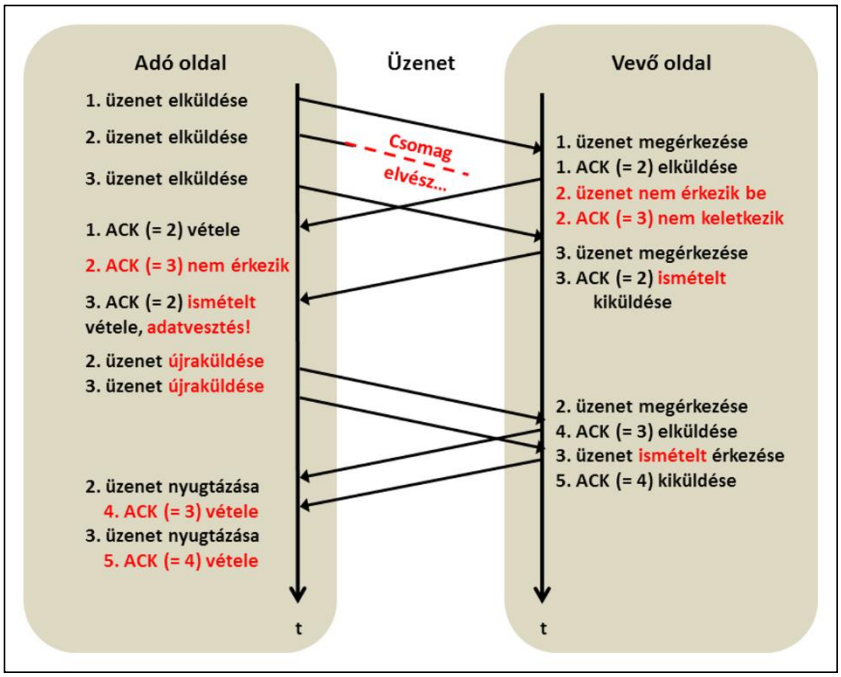
Az ablak kezdeti méretét – a kapcsolat kiépítése idején – a vevő határozza meg (window advertisement). Valamennyi kiküldött nyugtával a vevő újra és újra frissítheti az ablak méretét, ahogy ezt a vételi oldalon lévő szabad pufferek száma megköveteli. Ha a vételi oldalon lassú a beérkezett adatok feldolgozása, az a pufferek megteléséhez vezet, amit a vevő az adó felé küldött – 0 méretű ablakra figyelmeztető – üzenettel jelez, amivel eléri az adó leállítását (mivel az Aktuális pointer utoléri az Elküldhető pointert, vagyis az „*ablak kimerül*”). Ezt a megoldást folyamat-befolyásolásnak – flow control – nevezzük. Az ablak méretének helyes megválasztása nagyban befolyásolja a protokoll hatékonyságát.

A protokoll-gép a vételi oldalon hasonló ablakkal rendelkezik, amelyben összeállítja a bejövő adatokat, és tárolja, melyeket nyugtázta már. A full-duplex kommunikáció miatt valójában mindkét oldalon két-két ablak van a független kétirányú működés érdekében.

Hibajavítás visszalépés alkalmazásával

Ha az adó szerepét játszó protokoll-gép – a vevőtől érkező nyugták értelmezésével – felismeri, hogy a vevő adatvesztést jelez, igyekszik a hibát kijavítani. Ennek érdekében az elveszett adatcsomagot – továbbá az azt követő valamennyi adatcsomagot is! – újra elküldi. Erre készülve az adó a már kiküldött, de még nyugtára váró adatokat mindaddig puffereiben tárolja, amíg azok hibátlan vételét igazoló nyugtát nem kap. A pozitív nyugta beérkezésekor azonban nyomban felszabadítja a nyugtázott csomagot tároló puffert, hiszen az adat megőrzésének felelősségét a vevő a nyugta kiküldésével átvette.

Átviteli hiba javítása visszalépéssel



Megjegyzés:

Egy megismételt sorszámú nyugta beérkezése adatvesztésre utal, ezért az adó visszalép, és az elveszett csomagtól ismételten a vonalra küldi a visszalépéstől kezdve valamennyi csomagot.

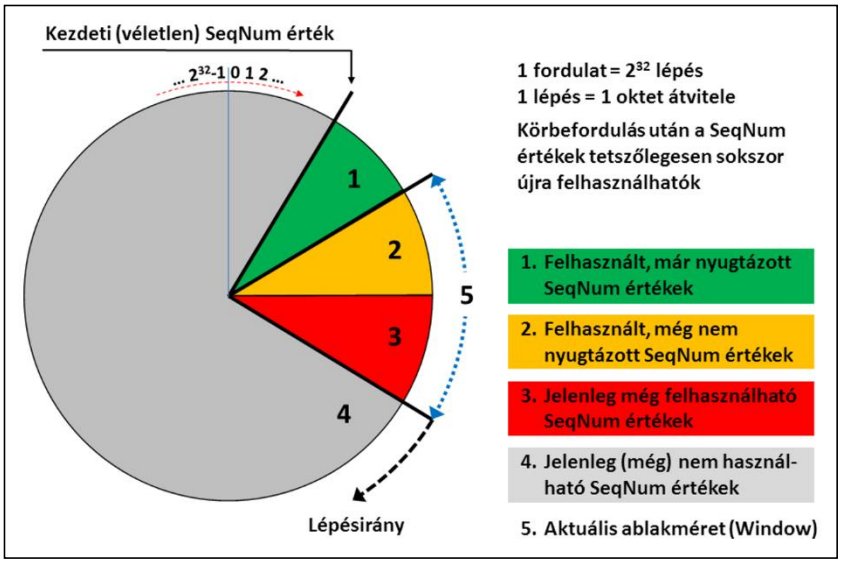
Amint az az ábrából kitűnik, ugyan csak a 2. üzenet veszett el, az adó – a visszalépés után – mégis mindazon csomagokat kiküldi, amelyek az elveszett – de most újra küldött – csomagot követik. Így a példa szerint a 3. üzenet feleslegesen kerül ismétlésre. Ez ugyan időveszteség, de egyszerűbbé teszi a visszalépést megvalósító algoritmust. Ezt a módszert nem-szelektív visszalépésként említik. (Modernebb TCP implementációk lehetővé teszik a szelektív ismétlés alkalmazását is. Ennek képességét a protokoll – TCP opció felhasználásával – jelzi a másik végpontnak, amely kihasználhatja e képességet, amennyiben ismeri annak módját.)

A csúszó ablakos protokoll minden vonalra küldött üzenetre külön időzítőt indít. Ezek valamelyikének lejárta az adott csomag elvesztésére utal, amely megint csak visszalépéses újraküldést eredményez. A visszalépés során az adó minden újraküldött csomaghoz tartozó – korábban indított – időzítőt leállít, és a csomagok kiküldésekor újakat indít.

Az eddig – egyszerűbb tárgyalás érdekében – írottakkal ellentétben, a TCP valójában nem a csomagokat sorszámozza be, e helyett sorszámként az adatcsomag első pozíciójában álló oktet virtuális csatornán – a kapcsolat kezdetétől – számított ofszetét használja sorszámként. Következésképpen a nyugta sorszáma sem a helyes sorrendben érkező következő csomag elvárt sorszámát, hanem a helyes sorrendben érkező következő oktet elvárt sorszámát jelzi.

A helyzet még ennél is bonyolultabb. Valójában a TCP a 0. ofszetű oktet sorszámaként egy – véletlen számként választott – 32 bites, előjeltelen számot – neve: Sequence Number – használ, a sorban következő oktetek sorszámaként pedig e véletlen szám értékét növeli oktetről-oktetre 1-el, majd az inkrementált értéket – a modulo 2^{32} aritmetika szabálya szerint – korrigálja. Az adó által választott véletlen számot a vevőnek is ismernie kell, ezért azt az adó – a kapcsolat kiépítése idején – tudatja a vevővel. (Ezt a műveletet természetesen a fordított irányban is el kell végezni!)

a TCP Sequence Number értelmezése



Megjegyzés:

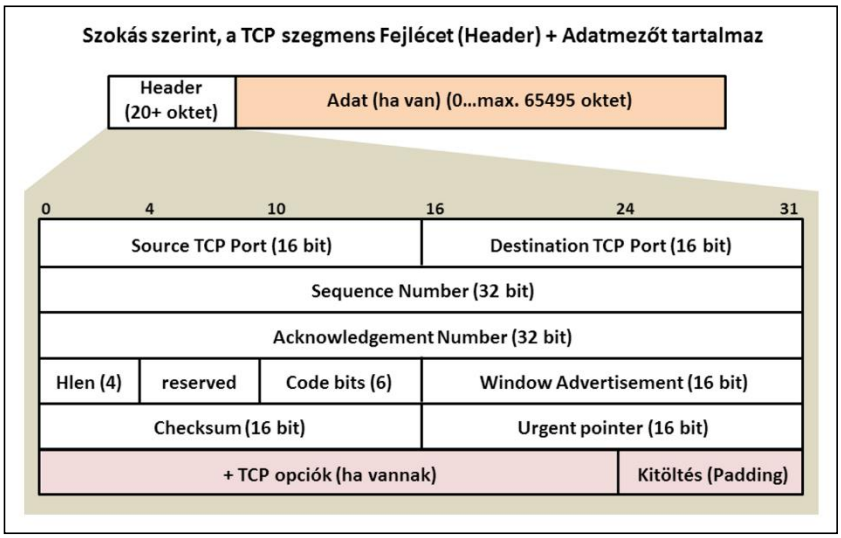
A véletlenszerű kezdőértéktől indulva a TCP egy-egy SeqNum értéket rendel minden átvitt oktethez. (Ezt egy körbeforgó vektorral szemléltetjük.)

A SeqNum értékek állapotai: [1] felhasznált (ablakból már kilépett), [2-3] felhasználható (épp az ablakban van), [4] most nem használható (még nincs az ablakban).

A TCP szegmens formátuma

Minden TCP forgalom (ami lehet: kapcsolat felépítése, adatok átvitele, nyugta/ablak méret hirdetmény küldése, kapcsolat lezárása) az alábbi szerkezetű szegmensben halad át az egyik végpontból a hálózaton keresztül a másik végpontba:

a TCP szegmens (szolgálati üzenet) formátuma



Megjegyzés:

A fejlécben állhat különféle TCP opció, és – szükség esetén – az opciók hosszát 32 bitre kiegészítő padding. A TCP szegmensből az adat akár hiányozhat is, ekkor a szegmens csak szolgálati üzenetet (ilyen pl. a nyugta) szállít.

Az egyes fejléc-mezők jelentése a következő:

Source Port = a szegmenst létrehozó process TCP port száma

Destination Port = a szegmens végcélját jelentő process TCP port száma

Sequence Number = a szegmensben szállított adat első oktetjének sorszáma

Acknowledgement Number = (ha ACK=1) azon oktet sorszáma az adatfolyamban, amelyet a vevő a legközelebb beérkező szegmensben elvár, ha az adatok átvitele hibátlan. Ez az ellenkező irányú adatfolyamra vonatkozik!

Hlen = a szegmens fejlécének hossza 32 bites egységekben

Code bits = az üzenet – a fejléc – tartalmára utaló jelzőbitek (flag)

Window Advertisement = a vételi oldalon rendelkezésre álló szabad puffer oktetben számolt mérete (a vevő által még tárolható oktetek száma). Az érték az ellenkező irányú adatfolyamra vonatkozik!

Checksum = a TCP fejléc + adatok helyes voltát ellenőrző összeg

Urgent pointer = (ha URG=1) a szegmensben szállított sürgős adat kezdőcíme

A Code mező bitjeinek jelentése (balról jobbra):

- **URG** = Urgent Pointer mező érvényes adattal feltöltve (sürgős adat szállítás)
- **ACK** = Acknowledgement mező érvényes adattal feltöltve (nyugta küldés)
- **PSH** = A szegmensben lévő adat push műveletet kér (azonnali átadás)
- **RST** = A kapcsolat megszakításának igénye (Reset) (protokoll-gép hiba!)
- **SYN** = Kapcsolat felépítés zajlik (Synchronize)
- **FIN** = Az adó az adatfolyam végére ért (nincs több adat az adott irányban)

A működést befolyásoló paraméterek cseréje, TCP opciók

Amint a fejléc felépítéséből kitűnik, hasonlatosan az IP-hez, a TCP is lehetővé teszi a működést befolyásoló kiegészítő paraméterek – opciók – használatát, és e paraméter értékek végpontok közti kicserélését. Az opciók többsége valóban opcionális, ritkán használt. Egy – a kapcsolat kiépítése idején alkalmazandó – *kötelező* opciót ismerünk, ez a Maximum Segment Size, MSS opció.

Az MSS opció segítségével – amely csak a TCP kapcsolat kiépítésekor alkalmazható, de akkor kötelező – a két kapcsolatba lépő protokoll-gép közli a másik féllel az általa feldolgozható (tárolható) szegmens oktetekben számolt maximális hosszát. Az érték jellemzően a közlő oldalán a TCP kapcsolatban felhasznált interface érvényes Maximum Receive Unit, MRU (más értelmezésben Maximum Transmit Unit, MTU) paramétere, pontosabban annál 2*20-szal (IP+TCP fejlécek) kisebb szám. Ha a partner ennél hosszabb szegmenst küld, az vagy nem érkezik meg a TCP protokoll-géphez, vagy nem tudja azt feldolgozni. Ha a felek nem egyeztetik az MSS értékét, annak alapértéke 536.

(Az MSS értékét az MTU/MRU mellett az adott gép puffer allokálási szabályai is befolyásolhatják. Ha egy beérkező szegmens tárolására egy fixméretű puffer szolgál, az MSS nem lehet nagyobb e puffer hosszánál.)

Ismert opciók: *Window Scale*, *Selective Acknowledgment Permitted*, *Selective Acknowledgment Data*, *Timestamp*. Ezek részletezésétől eltekintünk, referenciaként e segédlet végén található, a TCP-re vonatkozó RFC-ket felsoroló lista szolgál.

*A TCP **checksum** számítása*

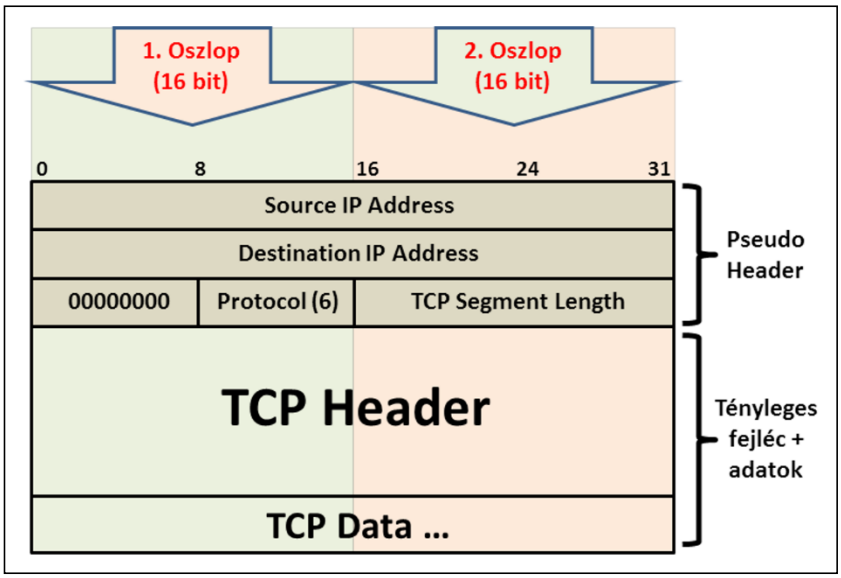
A TCP fejléc Checksum mezejének adó általi kitöltése, majd a vevő általi ellenőrzése kötelező feladat. Az ellenőrző összeg számításakor – az UDP-nél is alkalmazott – Pseudo Header, valamint a teljes TCP szegmens tartalmát figyelembe kell venni.

A Pseudo Header felhasználásának célja, hogy a szegmens esetleg hibás logikai kezelését – tartalmának protokoll-gép általi helytelen megváltoztatását, illetve a szegmens helytelen hálózati címre továbbítását – felismerjük. Mivel a Pseudo Header-t alkotó adatok nem a TCP szegmensből, hanem az azt a hálózatban szállító IP datagramból származnak, így az esetleges hibák felismerésének esélye növekszik. Végső soron azt mondhatjuk: az adó által kiszámított checksum érték vevő általi ellenőrzése csak akkor lehet sikeres, ha a tartalmában változatlan TCP szegmens valóban abba a gépbe jutott el, ahová azt a checksum kiszámításának idején szándékoztuk.

A Pseudo Header nevét azért kapta, mert valójában nem(!) létezik. Tartalma nem(!) része a TCP szegmensnek, nem(!) halad át a hálózaton, tartalmát csak(!) a checksum-számítás idején vesszük figyelembe. A Pseudo Header 12 oktet méretű, adat-tartalmát a következő ábrán láthatjuk.

Az ellenőrző összeg számítása során a protokoll-gép két 16 bites oszlopba rendezi a Pseudo Header + TCP fejléc + adatok tartalmát, majd modulo 2^{16} aritmetika szabályait követve összegzi azokat. (Ha az adat-oktetek száma páratlan, a számítás idejére kiegészíti azokat egy 0 értékű oktettel. A számítás idején a Checksum mező értékét 0-nak tekintjük.) Végezetül a kapott eredmény 1-es komplemensét vesszük, ez lesz a 16 bites Checksum kiszámított értéke.

a Pseudo Header felépítése és a checksum számítás módja



Megjegyzés:

A Pseudo Header a TCP szegmenstől független helyről származó (IP címek), és helyben kiszámított – a 3. sor adatai, (Prot.= 6 = TCP) – 12 oktetnyi adatot tartalmaz. Tartalma nem kerül a vonalra, csak az ellenőrző összeg számítását szolgálja.

*Kényszerített adatküldés, a **Push** funkció*

Az a TCP tulajdonság, hogy az átvitt adat belső tagolását nem ismeri, így arra nincs is figyelemmel, a felhasználások döntő többségében tökéletesen megfelelő. Ez ugyanakkor azt is jelenti, hogy nincs is módunk az oktetek átvitt monoton sorozatában határokat kijelölni. A TCP saját – valójában a hálózat folyton változó körülményeihez igazodó – szempontjaira figyelemmel dönti el, a puffereiben tárolt, továbbításra váró adatokat mekkora darabokra tördelve, mikor továbbítja a virtuális kapcsolat másik végpontjára. Az átvitel ütemezését ténylegesen a már említett csúszó ablakos algoritmus, illetve a hatékonyságot fenntartani igyekvő szabályok betartása végzi. Mindez oda vezet, hogy a TCP felhasználói képesek befolyásolni az adatok TCP-hez juttatásának ütemét és mennyiségét, de nincs befolyásuk az adatok vonali továbbításának tényleges ütemezésére.

Vannak azonban olyan helyzetek, amikor a befolyás hiánya holtponthoz vezető körülményeket teremt. A probléma mindaddig nem bukkan fel, amíg a TCP felhasználója képes az átviendő adatok folyamatos előállítására és TCP-hez juttatására. Ilyenkor a TCP időről-időre adatokkal tölt fel egy általa optimálisnak vélt méretű szegmenst, majd annak beteltekor azt a vonalra küldi, egy erre alkalmasnak tartott pillanatban.

De mi történik, ha a felhasználói program pillanatnyilag nem küld több adatot a TCP-hez? Ekkor a TCP beérkező adatra várva leáll, és nem küldi ki azt a szegmenst, amely most még az optimálisnál kevesebb adatot tárol. A gond akkor válik nyilvánvalóvá, ha rájövünk: a TCP nem fog újabb adatot kapni mindaddig, amíg a már megkapott adatokat el nem küldi. De nem küldi el azokat, amíg nem gyűlik össze megfelelő mennyiség… klasszikus holthelyzet!

Példaként nézzünk egy ilyen esetet! Egy távolban működő gép terminálunkról TCP kapcsolaton keresztül kap utasításokat. Gépeljünk be egy (rövid) parancsot, majd várjunk a gép válaszára… amely nem érkezik, hiszen parancsra vár, arra, amit a TCP nem vitt át a vonalon, mert még csak kevés karakter gyűlt össze a pufferben… Holtpont.

A megoldás az, ha a TCP mérlegelés nélkül a vonalra küldi – az akár csak egyetlen karaktert tároló – szegmenst. Ezt az elvárt működést érhetjük el a TCP *kényszerített adatküldés* – angolul: *Push* – néven említett funkciója kihasználásával.

Ha Push módú adatátvitelre utasítjuk a TCP protokoll-gépet, az a „*kényszerűen*” kezelt adatokat – vegyük észre: ezek mennyisége akár jelentős is lehet, több szegmenst is megtölthetnek(!) – további mérlegelés nélkül a vonalra küldi, de a felhasználói „*kényszer*” tényét azzal jelzi, hogy az ilyen szegmens – vagy szegmensek(!) – fejlécében a PSH flaget 1 értékre állítja.

Amikor az adatot fogadó másik végpont felismeri, hogy az adatok átvitele „*kényszerű*” volt (PSH = 1), azok hibátlan vétele után – az esetleg lehetséges további optimalizálás mellőzésével – késlekedés nélkül átadja az adatot a vételi oldalon a TCP kapcsolatot fenntartó feldolgozó programnak.

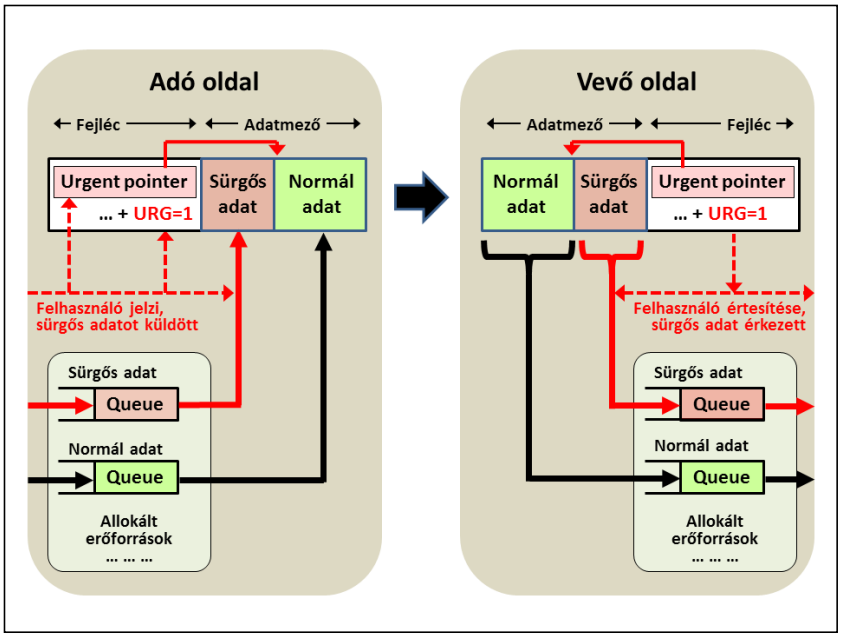
Fontos megjegyezni, hogy a Push funkció alkalmazása – miközben megoldja a korábban bemutatott holtpont kezelését – nem garantálja, hogy a „*kényszerűen”* küldött adatok átvitele olyan darabokban történik majd, ahogy azokat a feladó „*push-olta*”. A Push alkalmazása csak a várakozás nélküli továbbítást biztosítja, de a TCP már várakozó közönséges adatokkal együtt push-olhat „*kényszerűen*” küldött adatokat.

A TCP sürgős adatkezelése

A TCP – az adatfolyam szolgáltatás jellemzői között említettek értelmében – az átvitt adatok minden oktetét egyenlőként kezeli. Habár ez a FIFO (First In, First Out) viselkedés általában pont az, amit elvárunk, néha mégis szükség lenne bizonyos sürgős adatokat előnyben részesíteni. Ilyen eset lehet az, amikor meg akarjuk szakítani a már átadott adatok feldolgozását (pl. nyomtató leállítása). A leállító üzenet átvitele *sürgősebb*, mint a normál adatok továbbítása.

A TCP ismeri és kezeli a sürgős adatokat. Ha a felhasználói program sürgősnek minősített – pl. Abort parancsot tartalmazó – adatokat ad át, akkor a TCP ezeket egy új szegmens adatmezejének elejére másolja, a fejléc „*Urgent pointer*” mezőjébe a sürgős üzenet utolsó oktetét követő címre mutató értéket tölt, és a fejléc URG (Urgent) flagét 1 értékre állítja. Ha van még elküldésre váró normál adat, azt a sürgős üzenet mögé másolja – hogy elérje az optimális szegmensméretet –, majd a szegmenst késlekedés nélkül kiküldi. A kapcsolat másik végén az URG flag figyelmeztet a sürgős adat érkezésére, amelyet az ott dolgozó protokoll-gép elkülönít a közönséges adatoktól, és késlekedés nélkül átadja az üzenetet a feldolgozó programnak.

a TCP sürgős adatkezelése



Megjegyzés:

A felhasználói program sürgős adatküldés jelzésére a TCP új szegmenst nyit, amelyet az ábra szerint állít össze, és az URG=1 értékkel különböztet meg.

A szegmens a vevő oldalán is speciális kezelésben részesül az URG=1 értéknek köszönhetően. A sürgős adat átadása azonnal megtörténik, megelőzve a normál adatokat.

A leírtakból következően a felhasználói program tetszőleges időpontban, tetszőleges mennyiségű sürgős adatot adhat át a TCP-nek, csak jeleznie kell, hogy speciális kezelést igényel (más függvényt kell hívnia). A vevő oldalán a TCP jelzi a felhasználói programnak, hogy sürgős adatok érkeztek. A TCP addig nem ad át normál adatot feldolgozásra, amíg a felhasználó az összes sürgős adatot ki nem olvasta a pufferekből.

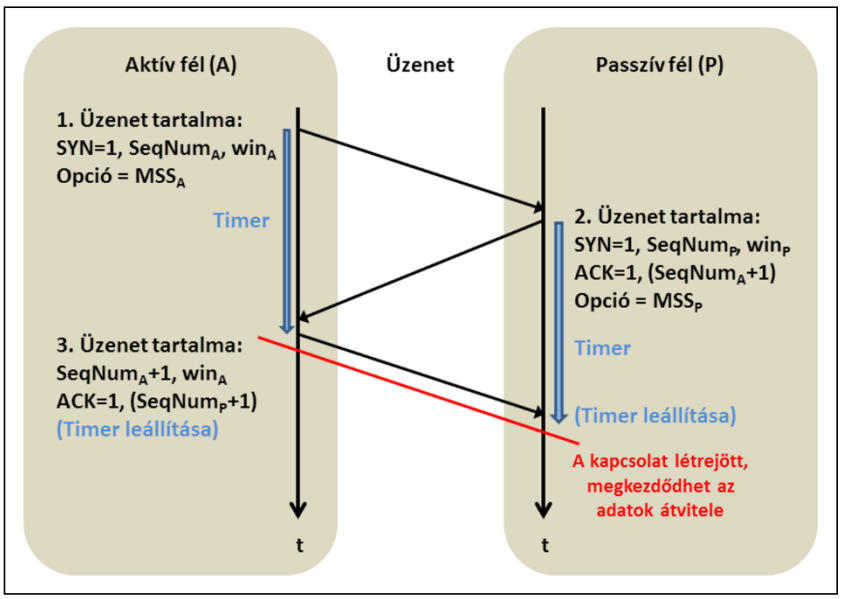
A leírt működés a hálózati szakzsargonban „*csatornán belüli jelzés*” – *In-band signaling* – néven ismert. Az elnevezés arra utal, hogy a speciális üzenetek ugyanazon a virtuális áramkörön haladnak át, mint a közönséges adatok, nincs elkülönített jelzési csatorna. (Ha lenne, akkor annak neve „*külön-csatornás jelzés*” – *Out-band signaling* – lenne.)

A TCP kapcsolatépítési folyamata

A TCP virtuális áramkörének létrehozását a majdani kapcsolat bármelyik végpontjáról kezdeményezhetjük. A kezdeményező az aktív fél, a másik a passzív fél. A kapcsolatépítés összesen három szolgálati üzenet felhasználásával zajlik le, ezért azt három-utas kézfogásnak – Three-way handshake – nevezik.

Az alábbi ábra ennek folyamatát mutatja be:

a TCP kapcsolatépítési folyamata (Three-way handshake)



Megjegyzés:

A kapcsolatépítés során az üzenetek adatot nem tartalmazó szegmensekben utaznak. Az ilyen szegmenst szolgálati üzenetnek nevezzük.

Amint látható, az aktív fél szolgálati üzenetével (1. üzenet) kezdeményezi a kapcsolat kiépítését. Az üzenetben a SYN – Synchronize – bit értéke 1. (Az üzenet kiküldésekor elindított timer feladata az elvárt nyugta elvesztése esetének érzékelése.) Az üzenetben az aktív fél néhány alapvető adatot küld a passzív szerepet játszó félnek. Ezek a következők:

- **SeqNum$_A$** – Az adatok sorszámozásához az aktív fél által véletlenszerűen választott sorszám (Sequence Number).
- **Win$_A$** – Az aktív fél vételi oldalán a passzív féltől majdan érkező adatok tárolására lefoglalt puffer oktetben számolt mérete (Window Advertisement, röviden Window).
- **MSS$_A$** – Az aktív fél a TCP fejlécben az MSS – Maximum Segment Size – kötelezően alkalmazott opció felhasználásával tudatja a passzív féllel, hogy maximálisan hány oktetnyi adatot tartalmazó szegmens vételére képes. (Szokásos értéke: 512…1460.)

A passzív fél a leírt tartalmú szolgálati üzenet vételekor értesül az aktív fél kapcsolatépítési szándékáról. Ha ezt elfogadja, nyugtázó szolgálati üzenetet (2. üzenet) küld, amelynek tartalma csaknem azonos a vett üzenettel (SYN=1), de kiegészül a nyugtával, amit az ACK (Acknowledgement) bit 1 értéke jelez. (Amint az aktív fél, úgy a passzív is timert indít.) Ebben az üzenetben átküldött adatok a következők:

- **SeqNum$_P$** – Az adatok sorszámozásához a passzív fél által véletlenszerűen választott sorszám (Sequence Number).
- **Win$_P$** – A passzív fél vételi oldalán az aktív féltől majdan érkező adatok tárolására lefoglalt puffer oktetben számolt mérete (Window Advertisement, röviden Window).
- **AckNum$_P$ + ACK=1** – jelzi, hogy a fejléc Acknowledgement Number mezője érvényes nyugtát tárol, mégpedig az aktív féltől kapott SeqNum$_A$ sorszám eggyel megnövelt értékét. (Úgy tekintjük, hogy a SYN üzenet felhasznált egy sorszámot.)
- **MSS$_P$** – A passzív fél a TCP fejlécben az MSS – Maximum Segment Size – kötelezően alkalmazott opció felhasználásával tudatja az aktív féllel, hogy maximálisan hány oktetnyi adatot tartalmazó szegmens vételére képes. (Szokásos értéke: 512…1460.)

Az aktív fél a nyugta beérkezésekor úgy tekinti, hogy a virtuális áramkör létrejött, de még nyugtáznia kell a passzív fél többcélú – kapcsolatépítő [SYN=1], és nyugtázó [ACK=1] – üzenetének vételét. Ezért maga is nyugtát küld (3. üzenet).

Ennek tartalma a következő:

- **$SeqNum_A$+1** – Az adatok sorszámozásához az aktív fél által véletlenszerűen választott sorszám 1-el megnövelt értéke, mivel a korábban kiküldött SYN=1 bit felhasznált egy sorszámot (Sequence Number).
- **Win_A** – Az aktív fél vételi oldalán a passzív féltől majdan érkező adatok tárolására lefoglalt puffer oktetben számolt – szükség szerint aktualizált – mérete (Window Advertisement, röviden Window).
- **$AckNum_A$ + ACK=1** – jelzi, hogy a fejléc Acknowledgement Number mezője érvényes nyugtát tárol, mégpedig a passzív féltől kapott $SeqNum_P$ sorszám eggyel megnövelt értékét. (Úgy tekintjük, hogy a SYN üzenet felhasznált egy sorszámot.)

A nyugta beérkezésekor immár a passzív fél is úgy tekinti, hogy a virtuális áramkör létrejött. Ettől kezdve a két végpont között megkezdődhet a kétirányú adatcsere.

Amennyiben a SYN = 1 üzenetekre időben nem érkezik nyugta (Timer lejár), a TCP sikertelennek értékeli a virtuális áramkör kiépítését, és azt vagy újrakezdi, vagy a kudarcról értesíti a kapcsolatépítést kezdeményező felhasználói programot.

A TCP kapcsolatbontási folyamata

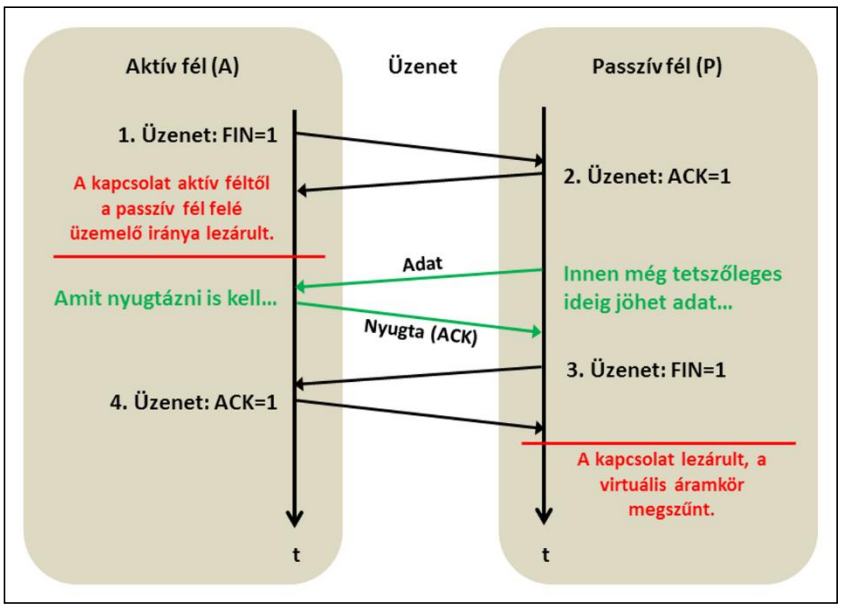
A TCP által kiépített virtuális áramkör kétirányú – duplex – adatátvitelt biztosít. Amikor a kapcsolatban álló felek valamelyike úgy dönt, hogy nem kíván több adatot a másik félhez küldeni, bontania kell a kapcsolatot. A kapcsolatbontás azonban nem eredményezi nyomban a virtuális áramkör elbontását, csak a bontást kezdeményező féltől a másik végpont felé üzemelő adatcsatorna lezárását váltja ki. Más szavakkal: A kapcsolatbontást kezdeményező fél több adatot nem küldhet ki.

A kapcsolat másik végpontján azonban még tetszőlegesen sok adat várakozhat kiküldésre. Ez meg is történhet, hiszen a kétirányú csatorna eddig csak az egyik irányba szakadt meg. Így aztán e féloldalas csatorna még tetszőleges ideig fenntartható, rajta adatok küldhetők át.

Fontos megjegyezni, hogy a szolgálati üzenetek kétirányú továbbításának lehetősége a kapcsolat végső bontásáig megmarad. Így aztán, habár az egyik irányba adatok már nem haladhatnak, szolgálati üzenetek azonban igen, amire szükség is van, hiszen a működő irányba elküldött adatok nyugtázását biztosítani kell, ez pedig ellenirányba haladó szolgálati csomagokat kíván.

A következő ábra a kapcsolatbontás két – időben elkülönült – lépését mutatja be:

a TCP kapcsolatbontási folyamata



Megjegyzés:

A virtuális csatorna két iránya időben egymástól függetlenül bontható el.

A megmaradt irányba adatok még átvihetők. Szolgálati üzenetek mindvégig mindkét irányba haladhatnak.

Amint az ábrán látható, a kapcsolatbontást kezdeményező aktív fél FIN=1 vezérlő bittel jelzi igényét, amit a passzív fél ACK=1 nyugtával igazol, de ez után még tovább folytathatja a nála várakozó adatok kiküldését. Egy későbbi időpontban aztán a passzív fél is befejezi az adatátvitelt, és – az előbbiekben leírtak szerint – maga is kezdeményezi a kapcsolatbontást. A leírtak szerint 2*2 lépésből álló szolgálati üzenetváltás végeztével a kapcsolat lezárul, a két kommunikáló végpont közti virtuális áramkör lebomlik.

A kapcsolat létének ellenőrzése, (Keep-alive)

A kiépített virtuális áramkört a TCP mindaddig fenntartja, amíg a felhasználói programtól a bontásra parancsot nem kap, illetve amíg olyan súlyos hibát nem észlel, amely után már értelmetlen a kapcsolat további fenntartása. (Rendkívüli kapcsolatbontás, lásd a következő fejezetet).

Egy kiépített TCP kapcsolat olykor percekig (órákig!) is tétlen maradhat, adatokat nem szállít. Ha nincs adatátvitel, nyugtát se kell küldeni… a vonal tetszhalott állapotot mutat. Ez azonban nem hiba, inaktivitásra hivatkozva nem bonthatjuk a kapcsolatot.

Vannak azonban olyan körülmények, amelyek a kapcsolat tényleges elhalását eredményezik, ráadásul a nélkül, hogy ennek bármilyen látható jele lenne. Gondoljunk csak arra az esetre, ha az egyik végpont összeomlik, avagy váratlanul kikapcsolják. A kapcsolat lebontása elmarad, adatátvitel se történik, de ezt a még működő fél nem tekint(het)i hibának. A helyzet a még működő végponton értékes erőforrások felesleges pazarlásához – szélsőséges esetben kimerüléséhez – vezet. (Gondoljuk meg: hányszor fordulhat elő ilyesmi egy éveken át folyamatosan üzemelő szerver életében. Ha megesik, a lefoglalt erőforrások majd újrainduláskor szabadulnak fel…)

Szükséges tehát, hogy az egyik végpont értesülhessen a másik állapotáról. Az ilyen vizsgálatot a szakzsargon többnyire „*heartbeat*” vagy „keep-alive” néven említi. (Lehetséges fordításuk „*szívverés*” illetve „*életjel*” lehet.)

A gond azonban az, hogy a TCP tervezői nem ügyeltek erre, így nincs olyan szabványos megoldás a végpont működőképességének ellenőrzésére, amelynek a másik félnél biztos meglétére döntést alapozhatunk. (Itt meg kell említeni, hogy a TCP protokollt leíró dokumentumok a mai napig ezt az ellenőrzést szükségtelennek ítélik, és az időközben kidolgozott megoldások alkalmazását is hangsúlyozottan opcionális lehetőségnek tekintik, amely alkalmazását kifejezetten csak szerver szerepet betöltő gépek esetében engedélyeznék.)

A vázolt probléma elvi megoldását a következőképp foglalhatjuk össze: Érjük el a túlfélen dolgozó protokoll-gépnél, hogy felhívásunkra életjelet adjon, de a vizsgálat ne zavarja meg a virtuális csatorna üzemét, és akkor is működhessen, ha – tetszőlegesen sokáig – nincs átküldhető adat. A kidolgozott ötletes megoldás az adott pillanatban utoljára felhasznált SeqNum érték újrahasznosításán alapul.

A vizsgálódó fél adatot nem tartalmazó szegmenst – vagyis szolgálati üzenetet – küld a másik félnek, választ remélve. Adatot azért nem küld, mert épp nincs. Ha lenne, a kiküldött adatra érkező nyugta egyúttal a másik fél működőképességét is nyugtázná, nem is lenne szükség a vizsgálatra.

A TCP fejlécben a SeqNum mezőt kötelesek vagyunk kitölteni. Rendesen itt a soron következő – még nem használt – SeqNum érték áll. Mivel adatot nem küldünk, ezért a túloldal – helyesen – úgy dönt, nincs mit nyugtázni, pointerei nem mozdulnak el, de nyugtát se küld, vagyis ellenőrzési kísérletünk sikertelen. A mérést fiktív adattal nem segíthetjük, mert azt a működőképes túloldal ugyan – célunkat elérve – nyugtázná, de a tényleges adatfolyam részének tekintve továbbadná a felhasználónak, ami az átvitt adatok eltorzítását jelentené.

Itt jön az ötlet: A fejlécbe írjuk be az aktuális SeqNum érték *előtti* számot, vagyis az utoljára használt SeqNum-ot, más szavakkal: használjuk fel azt újra. Ezt látva a vevő az eseményt sorrendhibás adat vételeként értelmezi. Adatot persze nem küldtünk – nincs mit az adatfolyamba illeszteni –, de a sorrendhiba hatására – a korábban leírt nyugtázási szabályok szerint – a vevő az elvárt SeqNum értéket közlő – vagyis az utolsó nyugtát megismétlő – figyelmeztetést küld vissza. Pontosan ezt vártuk! Némi praktikával rávettük a másik felet, hogy felfedje magát, most már tudjuk, hogy működőképes, így a TCP kapcsolat is az. Ha viszont nem érkezik nyugta, a kapcsolat már nem létezik, lebonthatjuk.

*Rendkívüli kapcsolatbontás (**Reset**)*

A TCP kapcsolatot a két végpontban működő TCP protokoll-gép együttes munkája biztosítja. A két gép egymással – szolgálati üzenetek formájában – kapcsolatot tart, így érve el kettőjük szinkronizált működését. Ritkán, de előfordulhat, hogy a gépek elvesztik a szinkront, ilyenkor valamelyik gép a másiktól érkező üzeneteket értelmetlennek ítéli (pl. a beérkező nyugtában olyan Sequence Number szerepel, ami nem is lett felhasználva). Ilyen esetben – más megoldás nem lévén – a szinkronvesztést előbb felismerő végpont rendkívüli kapcsolatbontást (Reset) kezdeményez. Ez egy olyan TCP fejléc – vagyis szolgálati üzenet – másik félhez küldésével történik meg, amelyben az RST bit értéke 1.

A virtuális kapcsolat azonnali bontását a felhasználói programból is elérhetjük. Ha a TCP protokoll-gép ilyen jelzést kap, ugyanúgy jár el, mint a szinkronizált működés elvesztésekor. Ekkor a kapcsolat elbontása mellett minden – az eddig fennálló kapcsolat érdekében – lefoglalt erőforrás (pl. pufferek) felszabadítása is megtörténik.

A TCP adatátvitelt optimalizáló algoritmusai

Az eddigiek alapján azt gondolhatnánk, az adó által a vevő felé küldött adatok mennyiségét csupán a továbbítandó adathalmaz mérete, és a vevő fogadókészsége (a rendelkezésére álló puffer mérete, másként: a vevő által hirdetet ablakméret) befolyásolja.

Valójában figyelembe kell venni a két végpont között elterülő hálózat pillanatnyi teljesítőképességét is, amiről – kezdetben – semmit se tudunk. Ráadásul a hálózat általunk tapasztalt teljesítményét jórészt nem is a mi forgalmunk, sokkal inkább a hálózatot velünk együtt használó – megszámlálhatatlanul sok – számítógép összegződő átviteli igénye befolyásolja. Ha a hálózat túlterhelt, az annyit tesz, hogy bizonyos pontokon olyan sok adat halmozódik fel, hogy azok továbbítására és/vagy átmeneti tárolására az ott üzemelő hálózati eszköz (router) már nem képes. Ekkor az adatátviteli teljesítmény jelentős csökkenését, a végpontok közti átvitel lelassulását – a késleltetés meghosszabbodását –, végül adatvesztést tapasztalunk. A helyzetet *torlódás* – angolul *congestion* – néven nevezzük. Nyilvánvaló, hogy a torlódás nem kívánt állapot, kialakulását lehetőleg el kell kerülni. A TCP adatátvitelt optimalizáló algoritmusai főképp e célt – a torlódás elkerülését – szolgálják.

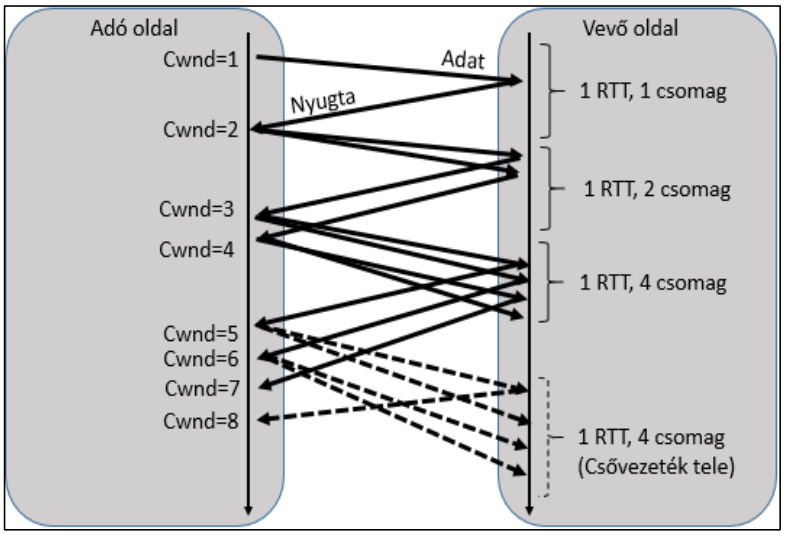
A Slow start algoritmus.

Másik fontos megfontolásunk az, hogy az AIMD-szabály szerint nagyon sok időt fog igénybe venni gyors hálózatokon a megfelelő munkapont elérése, ha a torlódási ablak kis méretről indul. Vegyünk egy szerény hálózati útvonalat, amely 10 Mbit/s sebességgel és 100 ms körülfordulási idővel (RTT) rendelkezik. A megfelelő torlódási ablak mérete a sávszélesség-késleltetés szorzat, ami 1 Mbit vagy 100 darab 1250 bájtos csomag. Ha a torlódási ablak mérete 1 csomagról indul, és minden körülfordulás alatt 1 csomaggal nő, akkor 100 körülfordulási időbe, tehát 10 másodpercbe fog telni, mire az összeköttetés sebessége a kívánatos közelébe ér. Ez elég hosszú idő annak kivárására, hogy elérjük a megfelelő átviteli sebességet. Csökkenthetnénk ezt az időt, ha nagyobb kezdeti ablakkal indulnánk, mondjuk 50 csomaggal. De ez az ablakméret lassú vagy rövid adatkapcsolatokra hatalmas lenne. Ha a teljes ablakot egyszerre kihasználnánk, akkor a leírtak alapján torlódást okozna.

Jacobson e helyett a fenti problémák megoldására a lineáris és az exponenciális növekedés egyfajta keveréke mellett döntött. Amikor az összeköttetés felépült, a küldő a torlódási ablak kezdeti méretét egy kicsi, maximálisan 4 szegmens méretűre választja; a részleteket az RFC 3390 tartalmazza. A 4 szegmens használata a korábbi 1 szegmens használatát váltotta fel a tapasztalatokból kiindulva. A küldő ekkor elküldi a kezdeti ablakméretet. A csomagok nyugtázása egy körülfordulási időbe kerülnek. Az újraküldési időzítő lejárta előtt nyugtázott minden szegmens után a küldő egy szegmensnek megfelelő számú bájtot hozzáad a torlódási ablakhoz, továbbá, mivel azt a szegmenst nyugtázták, eggyel kevesebb csomag lesz a hálózatban. A végeredmény tehát az, hogy minden nyugtázott szegmens után két új szegmenst lehet elküldeni. A torlódási ablak minden körülfordulási idő után megduplázódik.

Ezt az algoritmust lassú kezdésnek vagy lassú kezdést biztosító algoritmusnak (slow start) hívják, de egyáltalán nem lassú – valójában exponenciális a növekedés – csupán az előző algoritmussal szemben az, ahol a teljes forgalomszabályozási ablakot egyszerre küldték el. A lassú kezdést a 13. ábra mutatja. Az első körülfordulási idő alatt a küldő mindössze egy csomagot ereszt a hálózatba (és a küldő is egy csomagot kap). A következő körülfordulási idő alatt két csomagot, a harmadik körülfordulási idő alatt pedig már négy csomagot küld.

13. ábra



Megjegyzés:

Lassú kezdést biztosító algoritmus, egy szegmens méretű, kezdeti torlódási ablakkal

A lassú kezdés kielégítően működik olyan adatkapcsolatok esetén, ahol különböző a sebesség és a körülfordulási idő, és a küldő adási sebességének a hálózati útvonal sebességéhez való illesztéséhez nyugta órajelet használ. Vessünk egy pillantást a 13. ábrára, és figyeljük meg a nyugták érkezését a vevőhöz! Amikor a küldő egy nyugtát kap, akkor megnöveli a torlódási ablak méretét eggyel, és azonnal két csomagot küld a hálózatba. (Az egyik csomag az eggyel növelés miatt; a másik a nyugtázott, és így a hálózatot elhagyó csomag helyett kerül elküldésre. Bármely időpillanatban a torlódási ablak mérete adja meg az összes nyugtázatlan csomag számát.) Ez a két csomag azonban nem feltétlenül érkezik meg pontosan olyan időközzel a vevőhöz, mint ahogy elküldték azokat. Vegyünk például egy küldőt egy 100 Mbit/s Ethernet-vonalon. Minden 1250 bájtos csomag elküldése 100 s-ot igényel. A csomagok közötti időköz tehát akár 100 s méretű is lehet. A helyzet akkor változik meg, ha ezek a csomagok valahol az útvonal mentén áthaladnak egy 1 Mbit/s sebességű ADSL-adatkapcsolaton. Most 10 ms időbe telik ugyanannak a csomagnak a továbbítása. Ez azt jelenti, hogy a minimális időköz a csomagok között százszorosára nőtt. Ez az időköz ilyen nagy marad, hacsak nem várják be egymást egy várakozási sorban valamely gyorsabb adatkapcsolaton.

A 13. ábra ezt a jelenséget az adatcsomagok vevőhöz érkezésénél a nyilak végei között egy kisebb távolság feltüntetésével ábrázolja. Ugyanez a távolság szerepel a nyugták elküldésénél, és így a nyugták küldőhöz való visszaérkezésénél is. Ha a hálózati útvonal lassú, akkor a nyugták lassan érkeznek (egy körülfordulásnyi idővel később). Ha a hálózati útvonal gyors, akkor a nyugták is hamar megérkeznek (ugyancsak egy körülfordulásnyi idővel később). A küldőnek csupán a nyugtaórajelet kell követnie az új csomagok kiküldésénél. A lassú kezdés pont ezt teszi.

Mivel a lassú kezdés exponenciális növekedéssel jár, ezért előbb vagy utóbb túl sok csomag kerül túl gyorsan a hálózatba. Amint ez megtörténik, a hálózatban található várakozási sorok betelnek, és a teli sorok következtében csomagok fognak elveszni. Ezek után a TCP-küldő időzítője lejár, amikor egy nyugta nem érkezik meg időben. A lassú kezdés túl gyors növekedésének a 13. ábrán is szemtanúi lehetünk. Három körülfordulási idő után négy csomag található a hálózatban. A négy csomag megérkezéséhez a vevőhöz egy egész körülfordulásnyi időre van szükség. Tehát a négycsomagnyi torlódási ablakméret éppen megfelelő ehhez az összeköttetéshez. Ahogy azonban ezeket a csomagokat is nyugtázzák, a lassú kezdés tovább növeli a torlódási ablakot, és az a következő körülfordulási idő végére eléri a nyolccsomagnyi méretet. Összesen csupán négy csomag jut el a vevőhöz egyetlen körülfordulási idő alatt, függetlenül attól, hány csomagot küldtünk. A hálózati csővezeték tehát tele van. Ha a küldő további csomagokat bocsát a hálózatba, azok beragadnak az

útválasztók várakozási soraiba, mivel nem lehet a küldő felé elég gyorsan továbbítani. A torlódás és a csomagvesztés pedig hamarosan bekövetkezik.

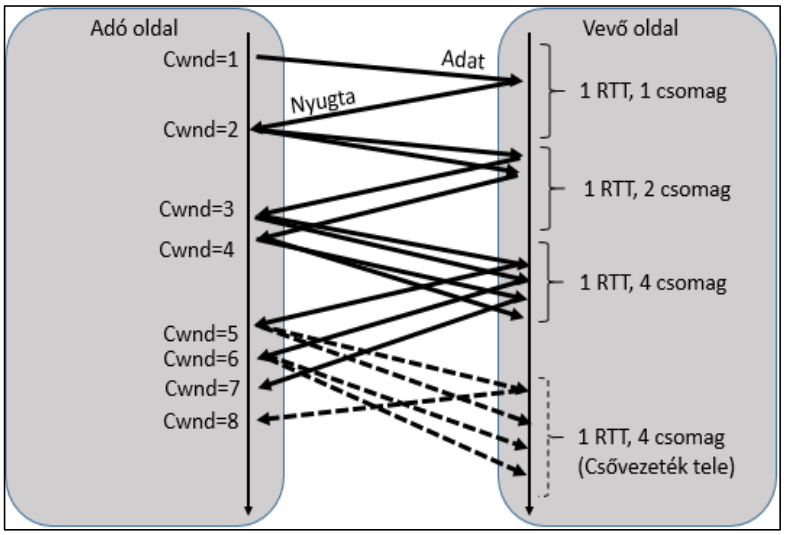
A küldő a lassú kezdés kordában tartására egy, a kapcsolathoz tartozó határt szab, amelyet a lassú kezdés küszöbértékének (slow start threshold) hívunk. Ennek értékét kezdetben önkényesen magasan tartjuk, a forgalomszabályozási ablak méretén, hogy az összeköttetés sebességét ne korlátozza. A TCP a lassú kezdés során a torlódási ablak méretét mindaddig növeli, amíg nem jár le az újraküldési időzítő, vagy a torlódási ablak át nem lépi ezt a küszöböt (vagy meg nem telik a vevő ablaka).

Amint a küldő – például az újraküldési időzítő lejárta miatt – felfedez egy csomagvesztést, a küszöböt a torlódási ablak méretének felére csökkenti, és a teljes folyamatot elölről kezdi. Az ötlet az, hogy az aktuális ablakméret túl nagy, mivel az előbbiekben torlódást okozott, és csak most, az időzítő lejártával vettük észre. Az ablak méretének a fele, amit már korábban is sikeresen alkalmaztunk, valószínűleg a legjobb becslése egy olyan torlódási ablaknak, amely közel van az útvonal kapacitásához, de még nem okoz csomagvesztést. A 13. ábrán látható példánkban a 8 csomagra növekedett torlódási ablakunk már okozhat csomagvesztést, míg a 4 csomag méretű ablak az előző körülfordulási idő alatt éppen a megfelelő nagyságú volt. A torlódási ablak méretét aztán az alacsony kezdeti értékére állítjuk, és visszatérünk a lassú kezdéshez.

Amint a lassú kezdés küszöbértékét átlépi, a TCP lassú kezdésről additív növekedésre vált. Ebben az üzemmódban a torlódási ablak méretét minden körülfordulási idő után eggyel növeli. A megvalósítás során a léptetés, ahogy a lassú kezdésnél is, általában minden nyugtázott szegmens után történik, nem pedig a körülfordulási időnként történő növeléssel. Legyen a torlódási ablak mérete cwnd és a legnagyobb lehetséges szegmensméret MSS. Egy gyakori megközelítés szerint a számú csomag minden nyugtájának megérkezése után a cwnd értékét értékkel növelik. A léptetésnek nem kell feltétlenül gyorsnak lenni. A megoldás lényege az, hogy a TCP-összeköttetés torlódási ablaka az idő nagy részét az optimális pont közelében töltse – ne legyen olyan kicsi, hogy lelassítsa az összeköttetést, de ne legyen olyan nagy sem, hogy torlódást okozzon.

Az additív növelést a 13. ábrán mutatjuk be a lassú kezdéssel megegyező esetben. A küldő torlódási ablaka minden körülfordulási idő végére annyit nő, hogy további csomagokat tudjon a hálózatra bocsátani. A lassú kezdéshez hasonlítva azonban a lineáris növekedés sokkal lassabb. Kisméretű torlódási ablakok esetén a különbség nem nagy (ahogy jelen esetben sem), de például az ablak 100 csomagnyi méretűre növelése esetén az időkülönbség már számottevő.

14. ábra



Megjegyzés:

A kezdeti, egy szegmens méretű torlódási ablak additív növelése

A teljesítmény növelésére egy másik eszközünk is van. Az eddigi megoldás gyenge pontja, hogy az újraküldési időzítő lejárását meg kell várnunk, ez pedig hosszú idő lehet, hiszen az időzítések meglehetősen nagyok. Ha egy csomag elvész, a vevő nem tudja nyugtázni, így a nyugtázott sorszámok nem változnak, és a küldő a torlódási ablak telítettsége miatt nem tud további csomagokat küldeni a hálózatba. Ez a helyzet sokáig eltarthat, egészen az újraküldési idő lejártáig. A csomagot ebben az esetben a TCP újraküldi, és újra lassú kezdésbe fog.

A küldő számára van egy gyorsabb módszer is a csomagvesztés megállapítására. Ahogy az elveszett csomag utáni csomagok beérkeznek a vevőhöz, a vevő nyugtákkal válaszol a küldőnek. Ezek a nyugták ugyanazt a nyugtasorszámot tartalmazzák, ezért ezeket nyugtamásolatoknak hívjuk. Amint a küldő egy ilyen nyugtamásolatot kap, feltételezheti, hogy a vevőhöz egy újabb csomag érkezett, de az elveszett csomag még mindig nem került elő.

Gyors újraküldés (fast retransmission)

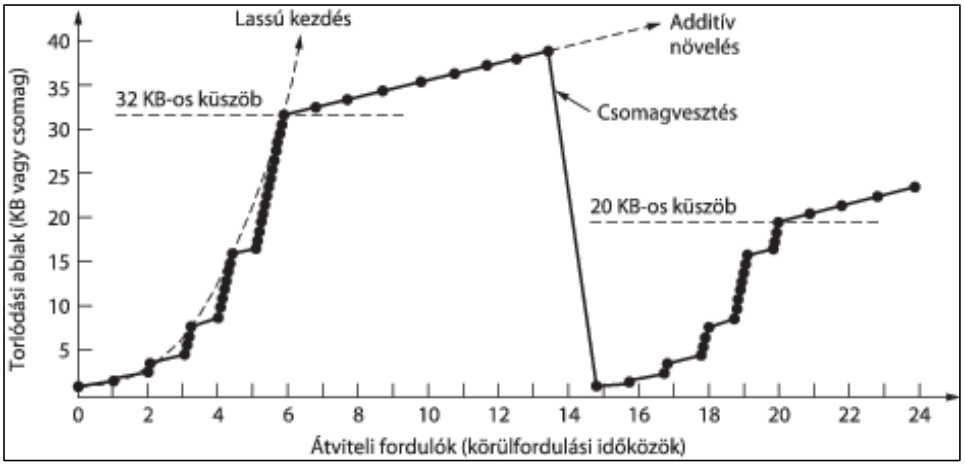
Mivel a csomagok különböző útvonalakat járhatnak be a hálózatban, nem feltétlenül kell sorrendben beérkezniük. Így előfordulhat, hogy csomagvesztés nélkül kapunk nyugtamásolatokat. Az interneten azonban ez nem túl gyakori. Tulajdonképpen a csomagok sorrendje akkor sem változik meg túlságosan, ha más útvonalon haladnak a hálózatban. Így a TCP, valamelyest önkényesen, azt feltételezi, hogy három nyugtamásolat vétele csomagvesztést jelent. Az elveszett csomagra a nyugták sorszámából következtethetünk, ugyanis a sorrendben következő csomag veszett el. Ez a csomag egyből újraküldhető anélkül, hogy az időzítő lejártát meg kellene várni.

Az ilyen heurisztikát gyors újraküldésnek (fast retransmission) nevezzük. Amint ez megtörténik, a lassú kezdés küszöbértékét ugyancsak a torlódási ablak pillanatnyi értékének felére állítjuk, ahogyan azt az időzítés túllépése esetén tennénk. A lassú kezdés a torlódási ablak egy csomag méretűre állításával kezdhető újra. Ezzel az ablakmérettel a TCP egy új csomagot küld minden körülfordulási idő után, amely az újraküldött csomag, és minden, a csomagvesztés előtt elküldött adat nyugtájának a visszaérkezéséhez szükséges.

Az eddigiekben felépített torlódáskezelő algoritmust a14. ábrán mutatjuk be. A TCP ilyen megvalósítását TCP Tahoe-nak hívjuk, ugyanis a 4.2BSD Tahoe 1988-as kiadásának része volt. A maximális szegmensméret 1 KB. Kezdetben a torlódási ablak mérete 64 KB, de egy időtúllépés következtében a küszöb értékét 32 KB-ra, a torlódási ablakot pedig 1 KB-ra állította a 0-dik menethez. A torlódási ablak mérete exponenciálisan nő, amíg el nem éri a küszöböt (32 KB).

Az ablakot minden nyugta beérkezésekor állítja, nem folytonosan, így egy diszkrét lépcsőmintázat alakul ki. Amint a küszöböt átlépte, a növekedés lineárisan folytatódik, minden körülfordulási-idő után egy szegmenssel nő.

15. ábra



Megjegyzés:

Lassú kezdést követő additív növekedés a TCP Tahoe-ban

A 13. körben az átvitel egy szerencsétlenség áldozata lett (sejthettük volna…) és egy csomag el is veszett a hálózatban. Három nyugtamásolat érkezése után a csomagvesztés kiderül, és ekkor a csomag újraküldésre kerül, a küszöb pedig az aktuális ablakméret felére csökken (azaz 40 KB-ról 20 KB-ra), és a lassú kezdés újraindul. Újra kezdve az egy csomagnyi méretű torlódási ablakkal, összesen egy körülfordulási időre van szükség, hogy az előzőleg elküldött, valamint az újraküldött csomagok elhagyják a hálózatot, és nyugtázásra kerüljenek. A torlódási ablak az előzőekhez hasonlóan növekszik mindaddig, amíg el nem éri a 20 KB méretű küszöböt. Attól kezdve a növekedés megint lineárissá alakul, és a folytatásban is e szerint növekszik, amíg nyugtamásolatok vagy időtúllépések csomagvesztést nem jeleznek (vagy a vevő ablaka meg nem telik).

A TCP Tahoe (amely megfelelő újraküldési időzítőket alkalmaz) egy működő torlódáskezelő algoritmust biztosított, és megoldotta a torlódási összeomlás problémáját. Jacobson azonban rájött, hogy készíthető ennél is jobb algoritmus. Amikor gyors újraküldés történik, az összeköttetés egy túl nagy torlódási ablakot használ, de a nyugtaórajel még mindig megfelelő. Minden esetben, amikor egy nyugtamásolat érkezik, valószínűleg egy újabb csomag hagyta el a hálózatot. Ha a nyugtamásolatotokat a hálózatban található csomagok megszámlálására használjuk, akkor lehetővé válik, hogy pár csomagnak megengedjük a hálózat elhagyását, és folytassuk az új csomagok küldését minden további nyugtamásolat esetén.

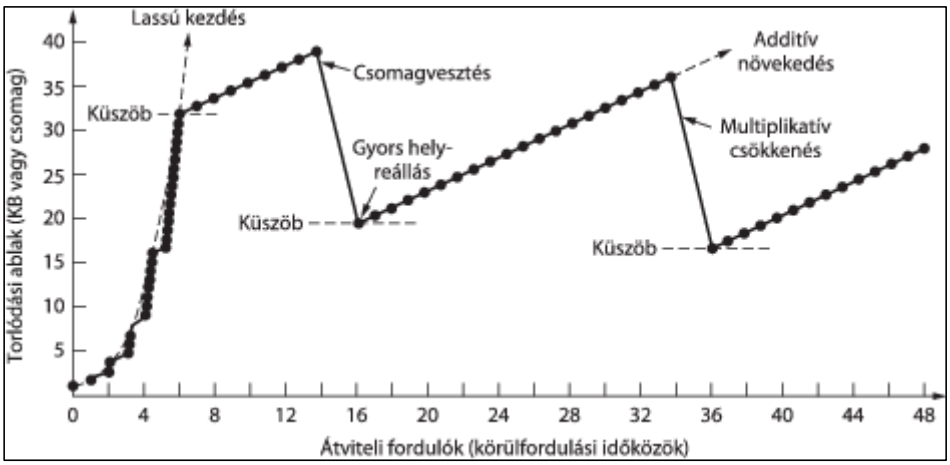
Gyors helyreállítás

A **gyors helyreállítás** (fast recovery) egy olyan heurisztikus algoritmus, amely ezt a viselkedést valósítja meg. Ez egy olyan ideiglenes állapot, ami megpróbálja a nyugtaórajelet tovább biztosítani egy olyan torlódási ablakkal, amelynek a mérete az új küszöbbel vagy a gyors újraküldés idején érvényes torlódási ablakméret felével egyenlő. Ennek elérése érdekében a nyugtamásolatokat folyamatosan számolja (beleértve azt a hármat is, amelyek a gyors újraküldést okozták) mindaddig, amíg a hálózatban lévő csomagok száma az új küszöb értékére nem esik. Ez körülbelül fél körülfordulásnyi időbe telik. Ettől kezdve minden egyes nyugtamásolat beérkezésekor egy új csomagot küldhetünk. A gyors újraküldés után egy körülfordulásnyi idővel az elveszett csomagra megérkezik a nyugta. Ekkor megszűnik a nyugtamásolatok özöne, és a gyors helyreállás üzemmód véget ér. A torlódási ablak mérete az új lassú kezdés küszöbértékére áll, és innentől lineárisan növekszik.

A heurisztika végkövetkeztetése az tehát, hogy a TCP elkerüli a lassú kezdést, kivéve, amikor az összeköttetést először felépítjük, vagy amikor időtúllépés következik be. Ez utóbbi még mindig megtörténhet abban az esetben, amikor több csomag is elvész, és a gyors újraküldés nem tudja kielégítően kijavítani a hibát. A lassú kezdések helyett egy működő összeköttetés torlódási ablakmérete **fűrészfog-mintázatot** alkot, additív növekedéssel (egy szegmenssel minden körülfordulási időnként), és multiplikatív csökkenéssel (egy körülfordulási idő alatt a felére csökken). Ez pont az AIMD-szabály, amit megkíséreltünk megvalósítani.

A fűrészfog-mintázatot a 16. ábra mutatja, amelyet a **TCP** Reno állított elő. A TCP Reno az 1990-ben kiadott 4.3BSD Reno része volt, és innen kapta a nevét. A TCP Reno gyakorlatilag a TCP Tahoe, gyors helyreállással megfűszerezve. A bevezető lassú kezdés után a torlódási ablak lineárisan növekszik mindaddig, amíg csomagvesztést nem észlel a nyugtamásolatoknak köszönhetően. Az elveszett csomagot újraküldi, és a gyors helyreállás segítségével az újraküldött csomag nyugtázásáig a nyugtaórajelet biztosítja. Innentől a torlódási ablak a növekedést az új lassú kezdés küszöbértékéről kezdi, nem pedig 1 szegmensről. Ez a viselkedés korlátlan ideig folytatódik, és az összeköttetés torlódási ablaka az idő nagy részét az optimális pont, tehát a sávszélesség-késleltetés szorzat közelében tölti.

16. ábra



Megjegyzés:

A TCP Reno gyors helyreállása és fűrészfog-mintázata

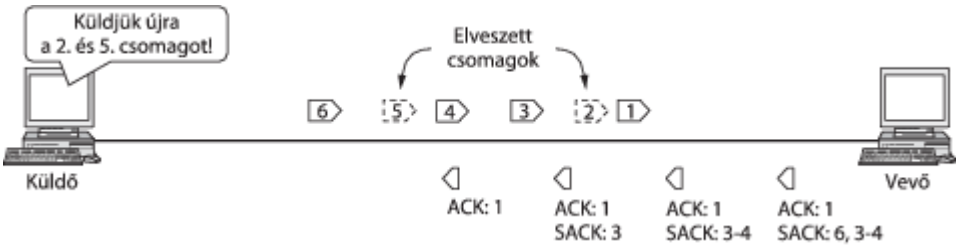
A TCP Reno a torlódásiablak-szabályozó megoldásaival több mint két évtizede meghatározza a TCP torlódáskezelésének alapjait. Az azóta eltelt évek során, a módszeren csupán apró módosításokat végeztek, mint például a kezdeti ablak paramétereinek megváltoztatását vagy különböző, nem egyértelmű részek eltávolítását. Némely továbbfejlesztés arra irányult, hogy egy ablakban történt két vagy több csomag elvesztését helyreállítsák. A TCP NewReno például egy újraküldés után a nyugtasorszámokat részlegesen előreléptei, hogy más csomagvesztéseket is megtaláljon és kijavítson [Hoe, 1996]. Az RFC 3782 részletesen ír a módszerről. A 90-es évek közepétől különböző olyan változatok jelentek meg, amelyek a leírt alapelveket követik, de valamelyest különböző vezérlési törvényeket alkalmaznak. Például a Linux a CUBIC TCP [Ha és mások, 2008], a Windows pedig a Compound TCP [Tan és mások, 2006] nevű változatot használja.

Szelektív nyugtázás

A TCP-megvalósításokra azonban két komoly változás is hatással volt. Az első szerint a TCP összetettsége főleg abból adódik, hogy a nyugtamásolatok tömkelegéből próbál következtetni arra, hogy mely csomagok érkeztek meg épségben, és mely csomagok vesztek el. A halmozott nyugtázás ilyen információt nem tartalmaz.

Az egyszerű megoldás a **SACK** (Selective ACKnowledgements – **szelektív nyugtázás**) használata, amely akár három bájtsorszám-intervallumot is felsorolhat, amelyek vétele sikeres volt. Ennek az információnak a segítségével a küldő sokkal közvetlenebbül el tudja dönteni, mely csomagokat kell újraküldeni, és a torlódási ablak megvalósításában is nagy segítséget nyújt, hogy a még úton levő csomagokat is nyomon lehet követni.

Amikor a küldő és a vevő felépítenek egy összeköttetést, mindketten elküldik a *SACK engedélyezve* (*SACK* permitted) TCP-opciót, amellyel jelzik, hogy a szelektív nyugtákat megértik. Amint a kapcsolaton a SACK engedélyezve van, akkor az a 18. ábrán látható módon működik. A vevő a TCP *Nyugtaszám* mezőjét a szokásos módon használja, tehát a legmagasabb, sorrendben beérkezett bájt halmozott nyugtázására. Ha a 3. csomagot kapja, ami nem a sorrendben következő csomag (mert a 2 csomag elveszett), a fogadott adatra egy *SACK opciót* küld, az 1 csomagra vonatkozó halmozott nyugta (-másolat) mellett. A *SACK opció* azokat a bájtsorszám-intervallumokat tartalmazza, amely adatokat sikeresen vett a vevő a halmozott nyugtában közölt sorszámon felül. Az első intervallum azt a csomagot jelenti, amely a nyugtamásolatot okozta, a többi jelenlévő intervallum pedig régebbi blokkok nyugtája. Általában legfeljebb három intervallumot használnak. Mire a hatodik csomag megérkezik, már két SACK bájtsorszám-intervallum jelzi, hogy a 6. és a 3-tól 4-ig terjedő csomagok sikeresen megérkeztek. Ez kiegészül az 1. csomagig terjedő halmozott nyugtával. Minden egyes beérkezett *SACK opció* lehetővé teszi a küldőnek, hogy meghatározza az újraküldendő csomagokat. Ebben az esetben a 2. és 5. csomag újraküldése lenne a jó ötlet.



17. ábra - Szelektív nyugtázás

A SACK szigorúan csak ajánlást nyújt. A tényleges csomagvesztés ténye ugyanúgy megállapítható nyugtamásolatokkal és a torlódási ablak paramétereinek állításával, mint az eddigiekben. SACK használatával azonban a TCP hamarabb helyreállíthatja az adatátvitelt olyan esetekben, amikor nagyjából egy időben több csomag is elveszett, mivel a TCP-küldő pontosan tudja, mely csomagok nem jutottak el a vevőig. A SACK manapság széles körben elterjedt. Működését az RFC 2883 írja le, a SACK-ot használó TCP-torlódáskezelést pedig az RFC 3517.

A másik komolyabb változás az ECN (Explicit Congestion Notification – explicit torlódásjelzés) használata a torlódás jelzésére a csomagvesztésen felül. Az ECN az IP-rétegbeli megoldás az állomások értesítésére, hogy a hálózatban torlódás található. Segítségével a TCP-vevő torlódási jelzést kaphat az IP-rétegtől.

Egy TCP-kapcsolat akkor használ ECN-t, ha mind a küldő, mind a vevő ezt jelezte az összeköttetés felépítésekor az *ECE* és *CWR* bitek beállításával. ECN használata esetén minden TCP-szegmenst megjelölnek az IP-fejlécben annak jelzésére, hogy az képes ECN-jelzések szállítására. Az ECN-t támogató útválasztók minden, ECN-jelzést szállítani képes csomagban jelzik, ha torlódás közeleg ahelyett, hogy a csomagot egyszerűen eldobnák annak bekövetkezte után.

A TCP-vevő a csomagban az ECN torlódási jelzést észlelve az *ECE* (ECN Echo) bit segítségével jelzi a küldőnek, hogy a csomagjai torlódást tapasztaltak. A küldő a vevő felé a jelzés vételét a *CWR* (Congestion Window Reduced – torlódási ablak mérete csökkentve) bit segítségével közli.

A TCP-küldő ezekre a torlódási jelzésekre pontosan úgy reagál, mint ahogy a nyugtamásolatokkal megállapított csomagvesztésekre. A helyzet azonban sokkal kellemesebb, hiszen a torlódást felfedezték, és egyetlen csomagnak sem esett bántódása. Az ECN-t az RFC 3168 írja le. Az ECN mind a hoszt, mind az útválasztó támogatását igényli, ezért még nem túl elterjedt az interneten.

Részletesebb információkat a TCP-ben alkalmazott torlódáskezelő megoldásokról, és azok viselkedéséről az RFC 5681 tartalmaz.

A Nagle algoritmus

Alapértelmezés szerint az adatküldési és adatfogadási műveletek egészen addig várakoznak, amíg nem tudják elküldeni az elküldeni kívánt adatokat, illetve nem érkezik adat a kommunikációs partnertől. Egyes alkalmazásoknál szükség lehet arra, hogy ezek a műveletek ne várakozzanak adatokra, hanem a művelet hívójához visszatérve jelezzék a várakozás szükségességének a tényét, hogy a program futhasson tovább.

A TCP-alapú összeköttetéseknél a rendelkezésre álló hálózati sávszélesség optimálisabb kihasználása érdekében lehetőség van a Nagle-féle algoritmus alkalmazására, ami nem engedi meg egynél több kevés felhasználói adatot tartalmazó nyugtázatlan csomag elküldését. Ennek alkalmazását az összeköttetés végpontjain a **setTcpNoDelay*()*** metódussal szabályozhatjuk: egyetlen logikai típusú argumentuma van, aminek IGAZ értéket átadva bekapcsoljuk a Nagle algoritmust, HAMIS értéket átadva pedig letiltjuk azt.

Bár a késleltetett nyugtázás csökkenti a hálózat vevő által okozott terhelését, a küldő a sok rövid csomag (például az egyetlen adatbájtot tartalmazó 41 bájtos csomagok) küldözgetésével még mindig pazarlóan gazdálkodik. Az ennek csökkentésére kidolgozott módszer a Nagle-féle algoritmus [Nagle, 1984] néven ismert. Nagle ötlete egyszerű: amikor a küldőhöz kis darabokban érkezik az adat, csak az elsőt továbbítja, a többit addig puffereli, amíg az elküldött darab nyugtája meg nem érkezik. Ezután a pufferben tárolt összes adatot egyetlen TCP-szegmensben elküldi, és újra kezdi a pufferelést, amíg a szegmens nyugtája meg nem érkezett. Így tehát csak egyetlen kis csomag lehet kinn egy időben. Ha a körülfordulási idő alatt az alkalmazás sok kisméretű adatot küld, akkor Nagle algoritmusa a sok kis darabot egy szegmensbe rakja, jelentősen csökkentve a használt sávszélességet. Az algoritmus továbbá kimondja, hogy egy új szegmenst kell küldeni, ha elég adat csordogált be, hogy megtöltsön egy maximális méretű szegmenst.

A Nagle-féle algoritmus széles körben elterjedt a TCP-implementációkban, de némely esetben szerencsésebb kikapcsolni. Például az interneten futtatott interaktív játékok esetén a játékos tipikusan rövid, frissítő csomagok tömkelegét szeretné küldeni mihamarabb. A frissítéseket bevárva, majd löketszerűen továbbítva a játékélményt erősen lerontaná, és sok elégedetlen felhasználót eredményezne. Egy még alapvetőbb probléma is felbukkanhat, ha Nagle algoritmusát a késleltetett nyugtákkal használjuk, ugyanis ideiglenes holtpontok alakulhatnak ki: a vevő az adatokra vár, hogy a nyugtát ráültethesse, a küldő pedig a nyugtára vár, hogy több adatot küldjön. Ez az összeférhetetlenség késleltetheti a weboldalak letöltését.

A fenti problémák miatt Nagle algoritmusa kikapcsolható (ezt TCP_NODELAY opciónak nevezik). Mogul és Minshall [2001] tárgyalja a problémát, és más megoldásokat is ad.

Körülfordulási-idő paraméter (rtt)

Ismétlési idő vagy körülfordulási-idő (Round Trip Time-RTT): minden TCP szegmens elküldésekor elindul egy időzítő, amely követi, hogy a nyugta időben megérkezik - e.

Erre Leggyakoribb algoritmus: $RTT = \alpha * RTT + (1 - \alpha) * M$

Az α egy átlagoló tényező, mely meghatározza mekkora súlyt kapjon a régi érték. az M a mért válaszidő. Kiszámítható a körülfordulási idő szórása

$$D = \alpha * D + (1 - \alpha) * |RTT - M|$$

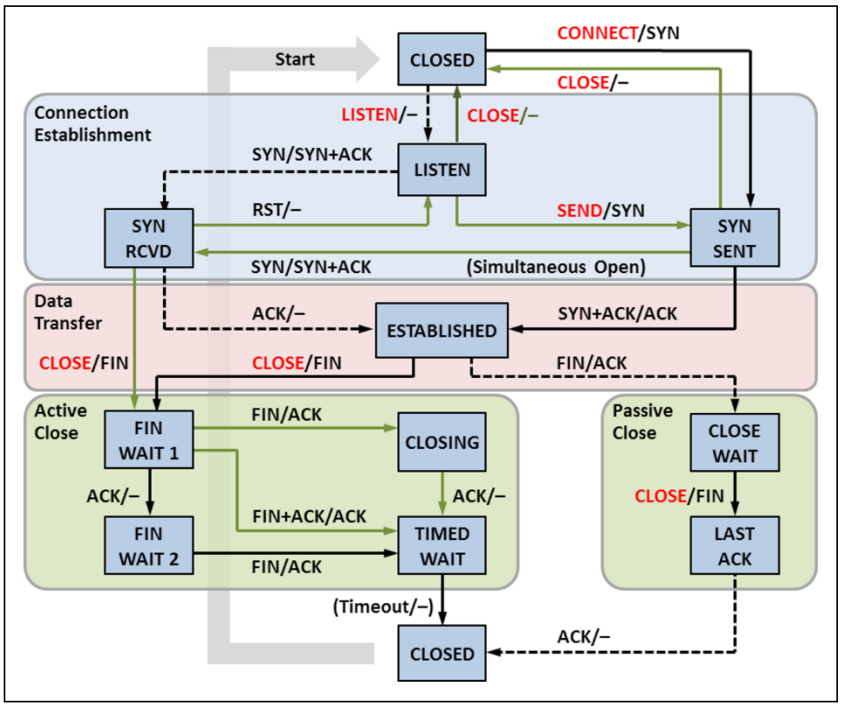
Az RTT meghatározható a szórás segítségével: $RTT = RTT + 4 * D$

Az elveszett vagy újraküldött csomagok véletlenszerűen módosíthatják az időzítőt, ezért a Karm féle javaslat szerint az újraküldött csomagok esetében nem figyeljük a nyugtákat, az időzítőt pedig a kétszeresére állítjuk.

A TCP állapotgép

A TCP protokoll működését egy determinisztikus, 11 állapotú állapotgép (*state machine*) valósítja meg. (Az állapotgépet nem más, mint egy alkalmas program.) A virtuális kapcsolat mindkét végén üzemel egy ilyen gép, amelyek szinkronizálást a két gép által a másikhoz küldött szolgálati közlemények biztosítják. Ezek a szolgálati közlemények jelentésüket a TCP szegmens fejlécében álló mezők megfelelő kitöltésével és elküldésével, illetve a másik végponton történő értelmezésével nyerik. Alább az állapotgép egyszerűsített állapot-átmenet ábrája látható:

a 11 állapotú TCP protokoll-gép egyszerűsített állapot-átmenet ábrája



Megjegyzés:

A kék négyzetek állapotokat jelölnek, bennük az állapot neve látható. A nyilak állapotváltást jeleznek, rajtuk a kiváltó külső esemény neve, a / után pedig a válasz szolgálati üzenet olvasható.

- A háttér színes területei a működés főbb fázisait – kapcsolatépítés, adatátvitel, kapcsolatbontás – szemléltetik.
- A protokoll-gép lehetséges állapotait és azok elnevezését a 11 darab kék négyzet jelzi. Alaphelyzetben az állapotgép CLOSED állapotban várakozik.

- A nyilak állapot-átmeneteket jeleznek. A fekete nyilak az általában bejárt utat, a zöldek a lehetséges, de ritkábban követet átmeneteket ábrázolják. A folytonos fekete nyilak az aktív szerepet játszó végpont lépéseit, a szaggatott nyilak a passzív szereplő útját mutatják.
- A nyilak mellett álló feliratok az állapotváltást kiváltó külső eseményt és – a / jel után – az erre adott választ szemléltetik. Ahol /– áll, ott csak állapotváltás történik, szolgálati üzenet nem keletkezik. Külső esemény lehet a TCP kapcsolatot felhasználó programtól érkező utasítás (piros parancsnevek), illetve a virtuális áramkör másik protokoll-gépe által küldött szolgálati üzenet (fekete üzenetek).

(Habár az ábra meglehetősen összetett, azt mégis egyszerűsítettnek kell minősíteni, mert pl. az időzítőkkel kapcsolatos belső események kezelését nem ábrázoltuk.)

Az állapotgép pontos működése a TCP-re vonatkozó RFC-k tanulmányozásával ismerhető meg. Bármely további részlet tisztázása is a megfelelő RFC ismeretét igényli. A TCP működését a következő RFC-k együttese írja le:

- Kapcsolódó standardok: STD 2, STD 3, STD 7
- Alapvető RFC-k: RFC 793, RFC 896, RFC 1122
- Számos további RFC foglalkozik még a TCP működésével. Ezek naprakész listája az RFC indexből ismerhető meg. Az index – csakúgy, mint valamennyi RFC és egyéb fontos adat – az *IANA – Internet Assigned Numbers Authority*, http://www.iana.org – honlapjáról tölthető le.

Röviden tehát, nem minden folyamat képes megfelelően működni adatvesztés esetén, ezért a megbízható szállítási szolgáltatásról a **Szállítási Vezérlési Protokoll (TCP)** gondoskodik, maximálisan kihasználva az IP tulajdonságait, de tőle függetlenül működik. A TCP megléte mentesíti az applikációkat, hogy maguknak keljen gondoskodni a hibamentes átvitelről. Mivel a TCP kapcsolatorientált, így a feladó és címzett között logikai kapcsolat jön létre. Az adó a sliding window érdekében 3 mutatót használ, ezek az adatfolyam offsetjei, azaz az elküldött oktetek számai. A Bal oldalon a legmagasabb offsetű oktet van, ez a nyugtázott keretek száma. Jobb oldalon a csúszóablak felső határa van. A középső részen a következőként elküldendő oktet áll. A bal és jobb oldali pointer közötti távolság az ablak mérete (nyugta nélkül elküldhető csomagok). A vevő a nyugtában jelzi, még hány oktet fogadására képes (ablakméret ajánlása).

TCP felépítése:

<table>
<tr><td colspan="5">Feladó port</td><td colspan="5">Címzett port</td></tr>
<tr><td colspan="10">Sorszám</td></tr>
<tr><td colspan="10">Ráültetett nyugta</td></tr>
<tr><td>Fejrész hossza</td><td></td><td>URG</td><td>ACK</td><td>EOM</td><td>RST</td><td>SYN</td><td>FIN</td><td colspan="2">ABLAK</td></tr>
<tr><td colspan="7">Ellenőrző összeg</td><td colspan="3">Sürgősségi mutató</td></tr>
<tr><td colspan="8">Opcionális mezők</td><td colspan="2">Padding</td></tr>
<tr><td colspan="10">Adatok</td></tr>
</table>

Hibajavítások:

Az adatátvitel során nem bízhatunk abban, hogy a vevő (hibátlanul) vette az elküldött adatokat. Ezért a szokásos eljárás a következő:

- Az adó - az alkalmazott protokoll által kijelölt szabályok szerint - keretekbe tördeli az átviendő információt, de eközben minden keret elejére egy sorszámot, végére pedig egy - az átvitt információ tartalmára jellemző - ellenőrző számot - CRC, Cyclic Redundancy Check - illeszt, majd elküldi az így kiegészített keretet, egyidőben elindítva egy időzítőt - timer - is.
- A vevő észlelve a keretet, annak tartalmából - az adóoldalival azonos szabályok szerint - ugyancsak elkészíti a CRC-t, majd összeveti azt a keretben érkezett CRC-vel. Ha a két érték eltér, ez átviteli hibát jelez. Ezután a vevő nyugtázó keretet - ACK, acknowledgement - küld az adónak, amelyben az adott sorszámú keret helyes/helytelen vételét jelzi.
- Az adó a beérkező nyugtából értesül a sikeres/sikertelen vételről. Ha hiba történt, az adó újra elküldi a sérült keretet. Ha a keret elküldésekor elindított időzítés úgy telik le, hogy nem érkezik vissza nyugta, ez azt jelenti, hogy a keret - vagy a nyugta! - az átvitel alatt elveszett. Ekkor az adó az adott keretet ismét elküldi.

Ismétlő módszerek

A megoldások több változtat ismert. Az elküldött adatok bitsorozatát bitek blokkjaira tördelik, és küldésnél minden blokkot egy előre megadott számszor újraküldenek. Például, úgy küldjük el a "1011" bitsorozatot, hogy minden bitnégyest háromszor küldünk el.

Tegyük fel, hogy elküldtük a "1011 1011 1011" sorozatot, amit "1010 1011 1011" sorozatnak veszünk. Mivel egy csoport eltér a másik kettőtől, megállapítható, hogy az átvitel hibás volt. Ez a megoldás nem túl hatékony; a hiba felderítése azon a feltételezésen alapul, hogy a vett csoportok eltérnek (például ha a "1010 1010 1010" csoportokat vettük, akkor ezt a módszer helyes átvitelnek detektálja).

Bár ez a megoldás igen egyszerű, mégis használatos: néhány eljárás a állomás számok küldésénél ezt használja.

Paritásellenőrző módszerek

Adott az elküldendő adatok folyama, amelyet bitek blokkjaira tördelnek, és minden egyes blokkban lévő 1 értékű bitet megszámolnak. A blokkhoz tartozó "paritás bit" értékét beállítják vagy törlik, attól függően, hogy a számlálás eredménye páros vagy páratlan volt. Ha az így ellenőrzött blokkok átlapolják egymást, akkor a paritás bitekkel még a hiba helye is behatárolható, és esetleg még javítható is, ha a hiba csak egy bitnél jelentkezik: ez a Hamming kódolás alapelve.

A paritás ellenőrző módszereknek vannak korlátai: egy paritás bit csak egyetlen egybites hiba felismerését garantálhatja. Ha két vagy több bit sérül, akkor a paritás ellenőrzés helyes eredményt adhat, annak ellenére, hogy az átvitel hibás volt.

Ciklikus redundancia-ellenőrzés (Cyclic redundancy check – CRC)

Sokkal átfogóbb hibafelismerő (és javító) módszert tesznek lehetővé a véges testek és a felettük értelmezett polinomok bizonyos tulajdonságainak kihasználása.

A ciklikus redundancia-ellenőrzés az adatblokkot egy polinom együtthatóinak tekinti, amelyet eloszt egy előre meghatározott, állandó polinommal.

Az osztás eredményének együtthatói alkotják a redundáns adatbiteket, a CRC-t.

- A vett adat ellenőrzéséhez elegendő megszorozni a CRC-t az előre meghatározott polinommal.
- Ha a szorzás eredménye a hasznos adat, akkor az adatok hiba nélkül érkeztek meg.
- Másik lehetséges ellenőrzési mód a CRC újbóli kiszámítása a hasznos bitekből, és a vett CRC-vel való összehasonlítása.

Forgalomvezérlés:

A számítógép hálózatokban különféle teljesítményű adatátviteli vonalak és számítógépek együttműködése valósul meg. Ez felveti annak problémáját, mit tegyünk, ha gyors adó lassú vevővel kényszerül együttműködni. Ekkor az adó előbb-utóbb elárasztaná adatcsomagokkal a vevőt. Ennek megakadályozására a vevő és az adó közötti alkalmas visszacsatolással forgalomvezérlést - flow control - kell alkalmazni, amellyel a vevő az adó sebességét a számára elfogadható mértékre csökkentheti. A hibajavításhoz hasonlóan számos különféle módszer ismert, ezek elve azonban hasonló: az adó csak korlátozott számú keretet küldhet - ezt a korlátot gyakran ablaknak, window nevezik - a vevő felé, és e korlát kimerülésekor le kell állnia mindaddig, amíg a vevőtől a további adást engedélyező jelzést nem kap. A különféle forgalomvezérlő módszerek erősen eltérő vonali hatékonyságot eredményezhetnek.

7. Fejezet

DNS

Az interneten használt osztott név adatbázis: a DNS (Domain Name Service) egy osztott, hierarchikus adatbázis, amely állandóan használatos: minden web lap letöltésnél, levél közvetítésnél szerepe van, nélküle megbénulna az Internet hálózat. A DNS egy rendszer, a keresőgépekben is ezen a néven találjuk meg. A DNS ugyanakkor egy szolgáltatás, vagy még inkább egy szolgálat. A DNS egyúttal protokoll is.

Eredeti szabványa: RFC 1035, amelynek később számos továbbfejlesztése jelent meg.

A DNS lehetővé teszi internetes erőforrások csoportjaihoz nevek hozzárendelését olyan módon, hogy az ne függjön az erőforrások fizikai helyétől. Így a világhálós (WWW) hiperlinkek, internetes kapcsolattartási adatok konzisztensek és állandóak maradhatnak akkor is, ha az internet útválasztási rendszerében változás történik, vagy a részt vevő mobileszközt használ. Az internetes tartománynevek további célja az egyszerűsítés, egy doménnevet (pl. www.example.com) sokkal könnyebb megjegyezni, mint egy IP-címet. A felhasználók így megjegyezhetik a számukra jelentést hordozó web- (URL) és emailcímeket, anélkül, hogy tudnák, a számítógép valójában hogyan éri el ezeket. A DNS-ben a doménnevek kiosztásának és az IP-címek hozzárendelésének a felelősségét delegálják; minden tartományhoz mérvadó névkiszolgáló (autoritatív névszerver) tartozik. A mérvadó névkiszolgálók felelősek a saját doménjeikért. Ezt a felelősséget tovább delegálhatják, így az al-doménekért más névkiszolgáló felelhet. Ez a mechanizmus áll a DNS elosztott és hibatűrő működése mögött, és ezért nem szükséges egyetlen központi címtárat fenntartani és állandóan frissíteni. A Domain Name System specifikálja az adatbázis technikai képességeit, emellett leírja az internet-protokollcsalád részét képező DNS protokollt, részletesen meghatározza a DNS-ben használt adatstruktúrákat és kommunikációt.

Tartománynévtér

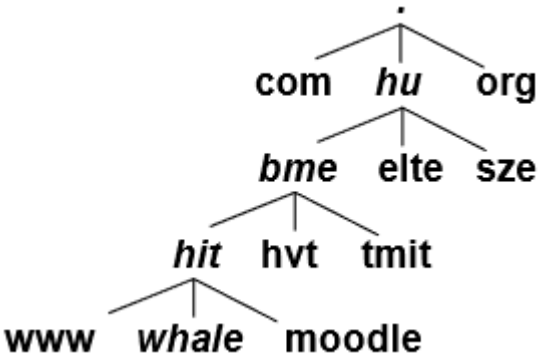
A DNS fordított fastruktúrájú hierarchiáját egymásba ágyazott tartományok (domének) alkotják, melyek szintjeit ponttal választják el egymástól, fontosságuk pedig jobbról balra haladva egyre csökkenő, pl. sub-b.sub-a.example.com. A fa minden leveléhez vagy csomópontjához nulla vagy több, a hozzá tartozó tartomány információit tároló erőforrásrekord tartozik. A fa adminisztratív egységekre, zónákra van osztva, a gyökérzónától kezdődően. Egy-egy DNS-zóna a fa összefüggő, önálló egységként kezelt része, állhat egyetlen doménből vagy tartozhat alá számos domén és aldomén, a kezelő által kiosztott adminisztrációs jogoktól függően. Egy zóna kezelője (földrajzi, topológiai vagy strukturális okokból) tovább delegálhatja a hozzá tartozó zóna egy része fölötti adminisztrációs jogát más feleknek. Ilyenkor a delegálással lényegében korlátozásmentes autonómiát ad át az allokált névtér fölött, a régi zóna adminisztrátorai, névkiszolgálói már nem mérvadóak az új zónára nézve. A tartománynevek szintaxisával kapcsolatos szabályokat az RFC 1035, az RFC 1123 és az RFC 2181 írja le. A tartománynév egy vagy több címke (label) láncolatából áll, melyeket pontok választanak el egymástól, pl. example.com. – a gyökérzónára utaló, leghátsó pont általában elhagyható, de hivatalosan az is része a teljes doménnévnek.

- A jobbszélső címke a legfelső szintű tartományt jelzi; például a www.example.com a com legfelső szintű tartományhoz tartozik.
- A tartományi hierarchia jobbról balra ereszkedik lefelé; két egymás melletti címke közül a baloldali a tőle jobbra eső címke egy aldoménjét határozza meg. Az előbbi példában az example a com domén aldoménje, a www pedig az example.com-é. Ez a hierarchia legfeljebb 127 szintű lehet.
- Egy-egy címke legfeljebb 63 oktet hosszúságú lehet, a teljes tartománynév a pontokkal együtt nem haladhatja meg a 253 oktetet. Ez a korlát a DNS belső, bináris reprezentációjában 255 oktet. A gyakorlatban egyes DNS-szoftvereknek további korlátaik lehetnek.
- A DNS protokoll önmagában nem szab korlátot a DNS-név címkéiben szereplő karaktereknek, elméletileg tetszőleges bináris karakterlánc előfordulhat benne. Az internet DNS-gyökérzóna tartományneveiben és a legtöbb aldomén nevében azonban egy preferált formátum és karakterkészlet használatos. A címkékben az ASCII karakterkészlet egy részhalmaza engedélyezett, ami az angol ábécé kis- és nagybetűiből, 0-9-ig a számokból és a kötőjelből áll. A tartománynevek kiértékelése kisbetű-nagybetű érzéketlen módon történik. A címkék nem kezdődhetnek vagy végződhetnek kötőjellel, és nem állhatnak csupa számból (bár létezik az interneten olyan tartománynév, ami nem tartja be ezt a szabályt).
- Az állomásnév (hosztnév, hostname) olyan tartománynév, amihez legalább egy IP-cím hozzá van rendelve. Például a www.example.com és az example.com tartománynevek egyben állomásnevek is, míg a com tartománynév nem az.

A tartománynévrendszer kliens-szerver modellt alkalmazó elosztott adatbázisra épül. Az adatbázis csomópontjait a tartományhoz irányuló kérdésekre válaszoló névkiszolgálók alkotják. Léteznek autoritatív és nem autoritatív (mérvadó/nem mérvadó) névkiszolgálók. Minden tartományhoz tartozik legalább egy mérvadó DNS-kiszolgáló, ami felelősként információt nyújt a tartományáról, illetve az alárendelt tartományok névkiszolgálóiról. A hierarchia csúcsát a gyökér-névkiszolgálók állnak, melyekhez a legfelső szintű tartományok feloldásakor kell fordulni. Az a névkiszolgáló tekinthető mérvadónak („autoritatívnak”) egy zónára nézve, ami olyan válaszokat ad vissza, amit az eredeti, „mérvadó” forrás (a tartomány adminisztrátora kézzel, vagy valamilyen dinamikus DNS-kezelő módszer segítségével) konfigurált, emiatt „biztonságosnak” tekinthetők – szemben a másod- vagy harmadkézből, egy más DNS-kiszolgáló lekérdezésével kapott válaszokkal.

A kizárólagosan mérvadó (authoritative-only) névkiszolgáló csak azokat a lekérdezéseket válaszolja meg, amire nézve mérvadó, tehát amit az adminisztrátor bekonfigurált. A mérvadó névkiszolgáló lehet „mester” (elsődleges) vagy „szolga” (másodlagos vagy tartalék) kiszolgáló. Az elsődleges kiszolgáló tárolja a zóna erőforrásrekordjainak eredeti kópiáit. A másodlagos kiszolgáló a DNS protokoll egy automatikus mechanizmusával, a zónaátvitellel éri el, hogy zónapéldánya megegyezzen az elsődleges példánnyal. Minden DNS-zónához tartoznia kell egy mérvadó névkiszolgáló-halmaznak, ami a szülő zóna NS rekordjaiból olvasható ki.

Felépítése:



Top Level Domain- TLD (Legfelső szint)

1. **Aldomain**
2. **Aldomain**
3. **Aldomain**

További információk: https://hu.wikipedia.org/wiki/Domain_Name_System#DNS_komponensei

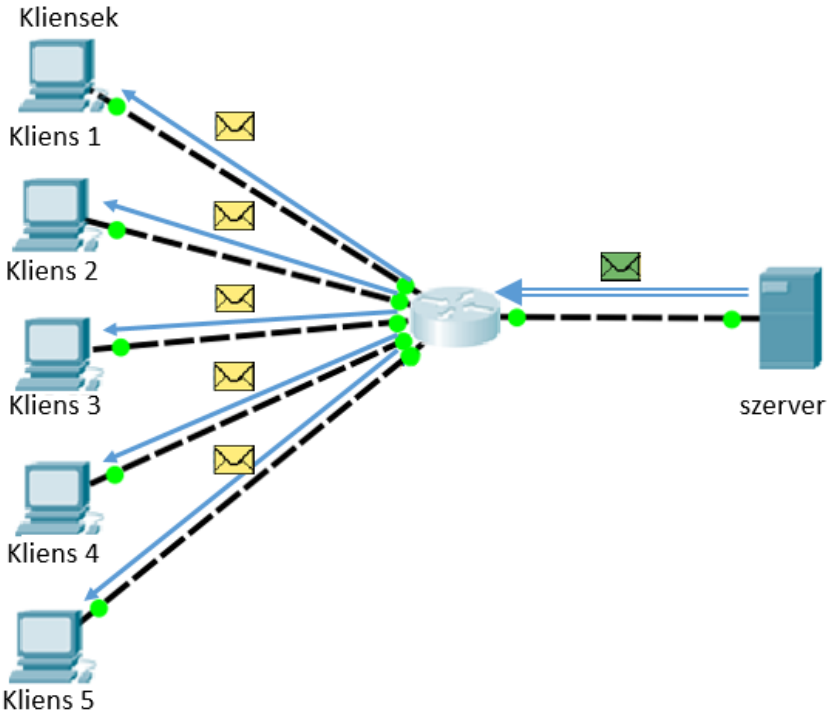
8. Fejezet

Multicast többescímzés az internetben

Egyre jobban terjednek a multimédia streaming alkalmazások a hálózaton, melyek a hagyományos pont-pont IP alapú összeköttetésekkel megvalósítva pazarolják a rendelkezésre álló sávszélességet, s nem alkalmasak a tömeges felhasználásra. Az IP multicast technika, amely a szolgáltatók használnak egy routolt Internet környezetben. Átgondolt, alapos tervezést, koordinációt és tesztelést igényel, mielőtt az éles hálózatukon használni tudnák.

A multicast egy kommunikációs forma, melyben egy adó (jelforrás) és tetszőleges számú vevő egy közös csatornát használva információt cserél. A legfontosabb tulajdonsága a pont-pont kommunikációval szemben, hogy az információmennyiség csak egy példányban áramlik át a hálózaton. Ezt a tulajdonságot elsősorban a multimédia alkalmazások tudják kihasználni, így az élő hang és videóközvetítések, a videókonferenciák, s az egyéb tartalomszóró megoldások. Az IP multicast lokális hálózatokon a legegyszerűbben megvalósítható: a szerver számítógép operációs rendszerének és az adást sugárzó szoftvernek képesnek kell lennie a multicast kommunikációra, pontosabban az **IGMP** protokoll használatára. Ez manapság minden modern TCP/IP implementáció része, így csak konfiguráció kérdése a hatékony használat. A legtöbb szoftver esetében ez csak a multicast címtartomány beállítását jelenti. Az összetett, routolt hálózatok esetében (mint az Internet) megoldandó a multicast forgalom pont-pont WAN linkeken keresztüli továbbítása. Ezt különböző multicast **routing** protokollok teszik lehetővé. Az első ilyen elterjedt protokoll a DVMRP volt, mely a világméretű MBONE hálózat kialakításánál fontos szerepet játszott. (Az MBONE az Interneten megvalósított multicast backbone, melynek különböző részeit un. tunnel-ek, azaz digitális átjátszó-alagutak kötik össze.) Mára egy korszerűbb protokoll, a **PIM (Protokoll Independent** Multicast**)** kezd elterjedni, ezt támogatják a Cisco routerek is. Természetesen lehetséges az átjárás is a különböző protokollok között.

A multicast működése



Megjegyzés:

Internet **Groum Membership** Protokoll (IGMP)

Az IGMP RFC 1112 az egyik legrégebbi és sokáig egyetlen a multicast-tal foglalkozó Internet szabvány, megvalósítása mára kötelező az állomásokban, az IPv6-ban pedig az ICMP szerves része. Nem broadcast link-ek esetén a multicast úgy valósítható meg, hogy minden a csoporthoz tartozó állomás számára egyesével elküldjük a multicast csomagot. Broadcast link-ek esetén azonban fontos, hogy a csomag csak egyszer haladjon végig a link-en. Az állomásoknak itt fel kell készülniük arra, hogy meghallják az összes D osztályú címmel feladott csomagot és kiszűrjék azokat, melyek nem nekik szólnak. Valamilyen módon azonban a router-ek tudomására kell hozni, hogy az adott link-en van-e valaki, aki tagja valamely multicast csoportnak, hogy azok egyáltalán körbeadják itt az annak a csoportnak szóló csomagokat. Erre való az IGMP. Két IGMP üzenet létezik, az egyik segítségével a router-ek kérdezik le a csoporttagságot, a másikkal pedig az állomások válaszolnak, egy-egy választ küldve minden csoporthoz, melynek tagjai. Mivel broadcast link-en mindegy, hogy hány tagja van az adott csoportnak, csak az fontos, hogy van-e vagy nincs, ha valamely állomás hallja, hogy más már jelezte tagságát a router-nek, egy olyan csoportra, aminek ő is tagja, akkor ő már nem jelez külön.

Multicast Routing

A multicast routing legegyszerűbb módja az árasztás. Az árasztás lényege az, hogy a bejövő csomagot minden interface-en továbbadjuk (kivéve azt, amelyiken kaptuk), csak arra kell figyelni, hogy ha egy csomagot már továbbítottunk, akkor ne továbbítsuk újra. Ehhez meg kell jegyezni a „néhány" legutoljára továbbított multicast csomagot, s ha valamelyik újra elérkezik hozzánk, nem továbbítjuk. A módszer hátránya éppen ez a lista, ami gyors hálózatokon meglehetősen hosszú lehet, másfelől azt ugyan garantálja, hogy egy router nem küld el egy csomagot kétszer, de azt nem, hogy nem is kapja meg kétszer, pedig ez is jó lenne. Sokkal hatékonyabb megoldás a **feszítőfa** módszere. Ezt használják a transparent bridge-k is (erről az Internetworking fejezetben volt szó), lényege az, hogy a hálózat fölött megállapítunk egy feszítőfát és a bejövő multicast csomagot e fa mentén továbbítjuk minden a fához tartozó interface-en, kivéve azon, amelyiken kaptuk. Minthogy a fa körmentes, egy csomag

kétszer senkihez sem jut el. Ráadásul igen kevés információt kell tárolnunk, csupán minden interface-hez egy bitet, hogy az adott interface tagja-e a fának vagy sem. A hátrányok között említeném, hogy a módszer mindenhova eljuttatja az összes multicast csomagot, függetlenül attól, hogy arrafele van-e tagja a csoportnak, másfelől pedig minden multicast forgalom a fa link-jein zajlik, más link-eken pedig egyáltalán nem, ami a hálózat egyenlőtlen terheltségéhez vezet. Kérdés még, hogy miként határozzuk meg a feszítőfát. Másik eljárás a **visszirányú továbbítás** (Reverse Path Forwarding, RPF). Ez explicit módon minden forráshoz külön fát épít egy tetszőleges routing protokoll táblázata alapján. A módszer előnye, hogy nem igényel semmi mást, mint egy teljesen tetszőleges routing protokoll táblázatát, sem külön memória, sem különösen nagy számítási teljesítmény nem kell hozzá. Ha a hálózat költségei szimmetrikusak, azaz a forrás felé ugyanannyi a költség, mint a forrás felől, akkor a multicast csomag minden célponthoz a legolcsóbb úton jut majd el, minden forrásból külön feszítőfa keletkezik. Emiatt a hálózat is egyenletesebben terhelt.

Multicast OSPF (MOSPF)

A legegyszerűbben a link-state protokollok alakíthatóak át multicast protokollokká, hisz ott a teljes adatbázis rendelkezésre áll. A MOSPF [RFC 1584] működése megegyezik az OSPF működésével, mindössze néhány kiegészítéssel bővült. Ugyanúgy, ahogy az OSPF router-ek közölik egymással, hogy támogatják-e a nem 0 TOS alapján való route-olást, azt is kiderítik egymásról, hogy támogatják-e a multicast-ot, így lehetséges egyszerre OSPF és MOSPF router-ek alkalmazása is, természetesen a multicast funkcionalitás így nem lesz teljes. Ha egy MOSPF router csomagot kap az F feladótól a C csoportra, akkor a topológiai adatbázis segítségével kiszámolja a célpontokhoz vezető legrövidebb utakat, majd a szükséges interface-eken keresztül továbbítja a csomagot. A topológiai adatbázisnak azonban csak azokat a részeit használja fel, melyeket MOSPF router-ek generáltak.

Ha így a terület 2 részre szakad, akkor a multicast forgalom nem tud eljutni az egyik félből a másikba. AZ MOSPF nem definiál az MBONE-hoz hasonló alagútrendszert, ez csak bonyolítaná a protokollt. Minthogy a MOSPF kiegészítés nem érinti a router hardware elemeit, egyszerű software cserével megoldható. A tervezők arra számítottak, hogy az átmeneti időszak rövid lesz. A csoporttagságok információit egy új link-state rekordban terjesztik a router-ek, melyben a link megbízott router-e felsorolja azokat a csoportokat, melyeknek van tagjuk az adott link-en. A területhatáron lévő router-ek ezeket a rekordokat is összegzik, és csak azt közlik a gerincre, hogy a területen milyen csoportoknak vannak tagjai. Az AS határán lévő router-ek nem terjeszthetik befelé az összes Internet csoportot, ez túl sok információ lenne. Az ilyen router-ek inkább automatikusan minden csoport tagjának vannak tekintve, így minden multicast csomagot megkapnak és tetszésük szerint továbbíthatják vagy elvethetik azokat. Hasonlóképpen a területhatáron lévő router-ek területük összes csoportjának tagjai, így a területen lévő összes csoportnak feladott multicast csomagot megkapják. Így a többi területen lévő csoportok listáját nem kell terjeszteni a területen belül. Ha két egyenlő költségű út létezik és ezek közül véletlenszerűen választanának a router-ek, mint az unicast esetben, előfordulhatna, hogy két router más-más utat választ, más feszítőfát kap eredményül, ami inkoherens döntéseket eredményezne. Ezért a protokoll tartalmaz egy algoritmust, ami alapján az egyenlő költségű utak közül a költségen kívüli szempontok alapján, egyértelműen lehet választani. A MOSPF nagyon sokat tudó protokoll, egyesek szerint túl sokat tudó. Hiszen minden a területen lévő csoporthoz, minden forrásból külön fát kell kiszámolni. Egy terület javasolt maximális mérete 200 link, ennyi forrás lehet, csoport akár még több is. Ez komoly számítási teljesítményt köthet le. A MOSPF fejlesztői szerint, ezeket a számításokat csak akkor végzi el a protokoll, amikor kell, tehát, ha az első csomag megérkezik az adott forrásból az adott csoportra. A számítások eredményét pedig cache-ben tárolják a router-ek, a cache akkor törlődik, ha már régen nem jött ugyanabból a forrásból ugyanabba a csoportba csomag, illetve topológiai változás esetén. Azok a router-ek melyeket nem érint az adott fa, sohasem fogják elvégezni a számításokat. Csupán az idő és a tapasztalat mondja majd meg, hogy a számítási teljesítmény elegendő lesz-e.

Protocol Independent Multicast (PIM)

Míg a MOSPF elsősorban az AS-en belüli multicast routing témakörét célozta meg és fő feladatának a legrövidebb út meglelését tekintette, addig a PIM inkább globális szinten keres megoldást és fő problémája a router-táblázatok mérete. A multicast csoporttól függően két verziója létezik, az egyik a sűrű (dense mode), a másik a ritka (sparse mode). Ezek a kifejezések azt takarják, hogy egy csoportnak mennyi tagja van az Interneten. Ha nagy az esélye annak, hogy egy adott területen van tagja a csoportnak, akkor az a csoport sűrű, ha pedig kicsi, akkor ritka.

Ennek a két szituációnak megfelelően két külön protokoll létezik.

- A sűrű PIM roppant egyszerű. Mivel a csoportnak mindenfelé vannak tagjai, nem okozunk túl sok fölösleges forgalmat, ha elárasztjuk a hálózatot a csomagokkal, éppen ezért az RPF tisztogatásokkal való kiegészítését használjuk. Ennek hibája, hogy a tiltakozások listáját időről időre el kell felejteni, hogy új tagok jelentkezése esetén hozzájuk is eljusson a csomag és a tiltakozás elmaradásával jelezzék, hogy kell nekik az adás. A lista elfelejtése azonban azzal jár, hogy számos tiltakozás újragenerálódik, ami nagy forgalmat okoz. Sűrű csoport esetén azonban nem olyan nagyot, hisz kevés a tiltakozó állomás. A sűrű PIM egyszerűbb, mint a DVMRP, mert nem működtet önálló unicast routing protokollt, adatait a létező protokoll táblázatából veszi. Ha egy router-ben kevés a memória, néhány régóta nem használt tiltakozási listát nyugodtan elfeledhet, ennek maximum a tiltakozások újraküldése lesz az eredménye. Ha két szomszédján keresztül egyenlő költségű utak vezetnek egy adott forráshoz, nemes egyszerűséggel úgy választ, hogy csak a nagyobb IP című szomszédjától fogad el csomagokat.
- A ritka PIM nem működhet úgy, mint a sűrű, tehát nem épülhet tiltakozásokra, hisz ehhez mindenkinek el kell juttatni az első csomagot, ami egy háromszemélyes videokonferencia esetén kimondottan felesleges a teljes Internetben.

A (ritka és sűrű) PIM üzenetek hordozói az IGMP csomagok, ám konkrét csomagformátumok még nem készültek el.

Forrásjegyzék

Felhasznált irodalom: https://hu.wikipedia.org, http://www.lan.hu/, http://www.cisco.com/, http://www.tankonyvtar.hu/hu/tartalom/tamop425/0005_24_szamitogepes_halozatok_scorm_03/338_az_ethernet_keretek_felptse.html, http://www.tankonyvtar.hu/hu/tartalom/tamop425/0046_szamitogep-halozatok_oktatasi_segedlet/ch02.html, http://www.inf.unideb.hu/kmitt/konvkmitt/szamitogep-halozatok_oktatasi_segedlet/book.xml.html, https:/www.szabilinux.hu/konya/konyv/, http://slideplayer.hu/slide/3035908/, http://uni-obuda.hu/users/racz.ervin/Adatatviteli_eszkozok.pdf, http://www.agr.unideb.hu/~agocs/informatics/01_h_history/h_jpte_internet_web/amirisc.ttk.pte.hu/network/AJ0403.htm, http://av.c3.hu/multicast/

Későbbiekben

Az analóg adatátvitel alapelvei Frekvencia/fázis/amplitúdó modulációs megoldások. Modemek célja, alapelvei, működési elvei. Modulációs, hibajavító, adattömörítő protokollok, modem szabványok, kompatibilitás. Kvázi digitális adatátvitel, V.90-es modemek. Az ADSL technika alapelvei. Digitális gerinchálózatok Városi és nagyterületű digitális adathálózatok működési elvei, felépítésük. Az SDH és SONET rendszerek áttekintése