Hálózati Rendszerek és Szolgáltatások Tanszék

# TCP/IP protocol stack vizsgálata

## Felkészülési segédlet

v1.0

A segédletet összeállította: Dr. Lencse Gábor, BME HIT, 2015.

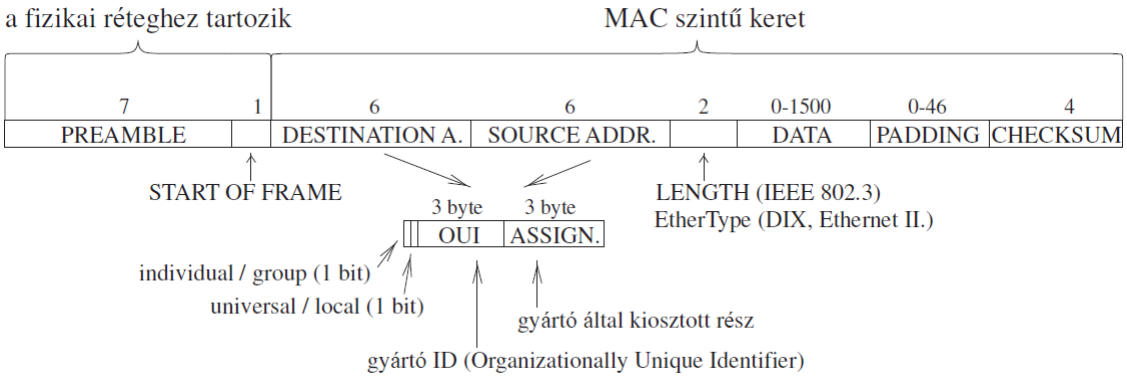
Tudnivalók:

- A felkészülési segédletet nem kell „kívülről megtanulni” a mérésre (nem lesz belőle beugró, viszont a mérésvezetők a felkészületlen hallgatót pótmérésre kötelezhetik), a célja az, hogy a hallgatók teljes mértékben megértsék a mérés elméleti hátterét. Az anyag szükség esetén a mérés közben is elővehető, referenciaként használható. A mérésre való felkészülés szempontjából a hallgatók többsége számára valószínűleg elegendő az, ha egyszer figyelmesen elolvassák és átgondolják a benne leírtakat. A mérés közben viszont erre már nem lesz idő, ezért erősen tanácsoljuk, hogy a mérés előtt szánjanak rá elegendő időt!
- A mérésre való felkészülés másik fontos része az, hogy a hallgatók a mérésre kipihenten és pontosan érkezzenek. Kérjük továbbá, hogy a mérés megkezdése előtt némítsák le a mobiltelefonjukat. ☺
- A mérések célja az, hogy a hallgatók gyakorlati módon ismerjék meg a tananyagot, az első mérésben konkrétan a *TCP/IP protocol stack működését*, illetve *készség szintjén elsajátítsák a Wireshark kezelését*, és *önállóan képesek legyenek a segítségével hálózati protokollok működését megvizsgálni*.
- A mérések anyaga publikus, de a feladatok előre történő megoldását nem javasoljuk: a mérés NEM vizsga, hanem lehetőség a hatékony tanulásra. Ezt felkészült és segítőkész mérésvezetők is támogatják azzal, hogy bármikor lehet tőlük kérdezni, segítséget kérni, ha valaki elakadna.
- A segédlet célja továbbá az is, hogy az előadás fóliák és az ajánlott jegyzet mellett a ZH-kra való felkészüléshez is segítséget nyújtson a tárgyalt témakörökben. A mérés és a felkészülési segédlet anyaga is a ZH-k anyagának részét képezik.
- Ebben a segédletben a következő témaköröket tárgyaljuk: Ethernet (csak részlegesen), IPv4 (ICMP-t is beleértve), TCP, UDP, ARP, DHCP, Wireshark (csak a szűrök röviden).
- A segédlet olvasójáról feltételezzük, hogy legalább a tárgy bevezető előadásait meghallgatta, így például ismeri az OSI 7 rétegű referenciamodelljét és a kapcsolódó alapfogalmakat. Ha valaki ezt valamiért elmulasztotta, kérjük, nézze át az előadás fóliáit vagy a tárgy ajánlott jegyzetét [1], amelyet felhasználtunk e segédlet elkészítéséhez.

# Tartalom

## Ethernet

Ma az Ethernet a helyi számítógép-hálózatok általánosan használt fizikai-adatkapcsolati szintű vezetékes megvalósítása. Ebben a segédletben csak a *keret*nek nevezett adategységének felépítésével foglalkozunk. Ennek felépítése az IEEE 802.3 szabványban és az elterjedten használt Ethernet II-ben egy mezőt illetően eltérő, az eltérést részletesen bemutatjuk. A keret felépítése az 1. ábra látható, melyből az első két mező a fizikai réteg része.




*1. ábra. Az IEEE 802.3 és az Ethernet II keretszerkezete*

Az egyes mezők megnevezése és feladata a következő:

**PREAMBLE** előtag, (bájtonként: 10101010)

Ez a bitsorozat arra szolgál, hogy a vevő az óráját az adó órájához szinkronizálhassa. (Például 10 Mbit/s sebességű hálózatnál 1 bit ideje: 100 ns, a 7 bájt 56 bitjének ideje tehát 5,6 µs.)

**START OF FRAME** keretkezdet határoló (10101011)

Az előtaghoz képest az utolsó bit 1-es értéke jelzi, hogy utána már a célállomás címe következik.

**DESTINATION ADDRESS** célcím

Bár az IEEE 802.3 megengedi a 2 bájtos hosszt is, gyakorlatilag mindig 6 bájt hosszú. Célszerű volt a többi mező elé tenni, hogy minden állomás minél előbb felismerhesse, hogy kinek szól a keret. (Felépítését külön tárgyaljuk.)

**SOURCE ADDRESS** forráscím

**LENGTH** adatmező hossza (IEEE 802.3) / **ETHERTYPE** típus (DIX, Ethernet II)

A DIX[1] Ethernet és annak v2.0 változata ezt a mezőt EtherType-nak nevezi, ami azt adja meg, hogy az adatmezőben milyen protokoll adategysége utazik. (Az IEEE szabvány itt az adatmező hosszát adja meg. A két megoldás akár egy hálózaton belül is használható, ugyanis az aktuális értékéből felismerhető, hogy hogyan kell értelmezni! Ha a mező értéke 0-1500 bájt közötti, akkor hossz, ha ettől eltérő, akkor EtherType.) A Wireshark csak a **Type** megnevezés használja erre a mezőre. Néhány gyakran használt érték jelentése: 0x0800: IPv4, 0x0806: ARP, 0x86DD: IPv6.

**DATA** adatok

Itt található beágyazva a felsőbb rétegbeli protokoll adategysége.

[1] Az IEEE 802.3 szabvány megszületése előtt a **D**EC, **I**ntel és **X**erox cégek által megalkotott „eredeti” Ethernet.

**PADDING** kitöltés

A keret minimális hossza 64 bájt, ha nincs annyi, ki kell egészíteni.

**CHECKSUM** ellenőrző összeg

Ha megegyezik a számított és a vett érték, akkor a keretet hibátlannak tekintjük, ellenkező esetben a keret sérült.

Az összes Ethernet hálózati kártya egyedi címmel rendelkezik. Ezt úgy valósítják meg, hogy a cím 6 bájtját két részre bontották. Az első 3 bájtot az IEEE osztja ki a gyártók részére.[2] A másik 3 bájtot pedig a gyártók osztják ki az általuk gyártott eszközöknek. Bár az OUI (Organizationally Unique Identifier) elvileg 24 bites, gyakorlatilag csak 22 bitet használnak a gyártók azonosítására, ugyanis az *első két bit* (lásd a lenti megjegyzést) más célt szolgál: az I/G bit (individual / group) 0 értéke azt jelenti, hogy valóban egy kártya *egyedi* címéről (unicast address) van szó, az 1 értéke esetén a cím egy *csoportcím* (multicast address). Ha az U/L (universal / local) bit értéke 0, akkor valóban univerzálisan (helyesebben: globálisan) adminisztrált címről van szó, aminek az egyedisége az imént említett módon biztosított, ha pedig a bit értéke 1, akkor a címet a rendszergazda osztotta ki (erre bizonyos protokollok esetén szükség lehet). Nagy jelentősége van még a *broadcast cím*nek, ami 48 darab 1-es bitből áll. Az ilyen címre küldött kereteket minden hálózati kártya veszi.

Megjegyzés: A címeket az 1. ábra az Ethernet által használt bitsorrendben adtuk meg*: legkisebb helyiértékű bit elöl* (least significant bit first). A Wireshark a matematikában és a köznapi életben is használt bitsorrendet alkalmazza: *legnagyobb helyiértékű bit elöl* (most significant bit first), vagyis a legkisebb helyiértékű bitet az első bájt jobb szélső bitjeként látjuk!

Egy hálózati interfész alapértelmezésben a következő kereteket veszi:

- azokat az egyedi (unicast) címre küldött kereteket, ahol célcím az interfész egyedi címe
- azokat a csoport (multicast) címre küldött kereteket, amely csoportnak az interfész tagja
- az összes üzenetszórás (broadcast) címre küldött keretet.

Ethernetnél a vételre vonatkozóan fontos a *promiscuous mode*: a hálózati interfész olyan működési módja, amikor válogatás nélkül minden keretet vesz. Ezt a működési módot használja a Wireshark is teljes hálózati a forgalom megfigyelésére.

Az Ethernetről részletesebb leírás található az [1] jegyzetben, illetve az Ethernetről szóló [2] szakkönyvben.

[2] A kiosztott azonosítók listája megtalálható: [3].

## Internet Protokollkészlet

Először tisztázunk néhány fogalmat. Az *internet* jelentése: összekapcsolt hálózat. Az *Internet* egyetlen világméretű nyilvános IP alapú internet. Az *intranet* internet technológiát használó intézményi belső hálózat. Az *internet protokollkészlet*be (internet protocol stack) tartozó legfontosabb protokollok:

**IP** (Internet Protocol, magyarul: internet protokoll[3])

Az IP egy nem megbízható *datagram*[4] szolgálatot biztosít a forrás- és a célgép között, függetlenül attól, hogy ezek hol helyezkednek el (azonos, szomszédos, vagy egymástól távoli hálózatban).

**TCP** (Transmission Control Protocol)

A TCP az IP-t felhasználva két végpont közötti megbízható, kétirányú bájtfolyam átvitelt biztosít. Ehhez a két végpont között kapcsolatot épít fel, amit az átvitel végén le kell bontani. A végpontokat (egy adott számítógépen belül) *portszám*mal azonosítja.

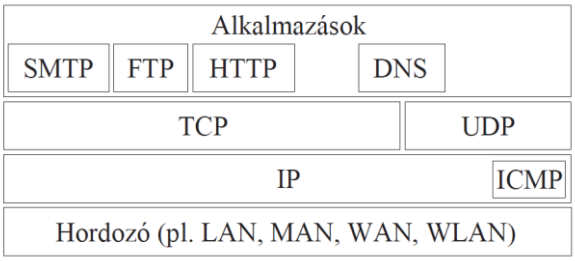
**UDP** (User Datagram Protocol)

Az UDP szintén az IP-re építve összeköttetés nélküli (itt nincs kapcsolatfelépítés, és kapcsolatbontás) végpontok közti nem megbízható datagram szolgáltatást, valamint ugyancsak portszám segítségével történő végpont-azonosítást nyújt a felhasználóknak.

**ICMP** (Internet Control Message Protocol)

Az ICMP az IP szolgálati közleményeit hordozza (szintén az IP-re építve) két IP-t használó állomás között. NEM önálló protokoll, hanem minden IP implementáció kötelező része!

A 2. ábrán megfigyelhetjük a felsorolt protokollok elhelyezkedését a TCP/IP modellben. A *hordozó réteg*re (link layer) épül az IP, erre pedig a TCP és az UDP. Az ICMP-t az általunk használt „egysíkos” számítástechnikai architektúrában az IP réteg részének tekintjük.[5] Először ezzel a négy protokollal ismerkedünk meg.




*2. ábra. A TCP/IP protokollcsalád*

Az internet protokollkészlet elemeinek (és sok más számítástechnikával, hálózattal kapcsolatos protokollnak) a definícióját az *RFC*-kben találjuk meg. Az RFC a *Request For Comments* rövidítése. Ezek a dokumentumok számos helyen elérhetők. Tanulmányozásukat HTML formázásban javasoljuk, mert ott megtalálható az esetleges újabb RFC-re való hivatkozás is, amit kiegészíti, esetleg érvényteleníti az adott RFC-t. Például egy jó kereső található a https://tools.ietf.org/html/ webcímen is. Az egyes protokolloknál a továbbiakban mindig megadjuk az azt definiáló RFC számát is. Mivel az

[3] Figyeljünk oda az eltérő helyesírásra!
[4] A csomag speciális neve az internet protokollkészletben.
[5] A távközlési architektúrában pedig egy külön úgynevezett *vezérlő* (control) síkon helyezkedik el.

RFC-k nem túl olvasmányosak, a témai iránt mélyebben érdeklődőknek ajánlunk néhány könyvet: [4]–[7].

## Hálózati bájtsorrend

Több bájtból álló adatok (például egész számok) esetén számítógépünk architektúrájától függően eltérő lehet az adatoknak a memóriában való tárolása. A bájtok sorrendjét illetően két ugyanolyan jó lehetőség van:

**LSB** Least Significant Byte first

A legkisebb helyiértékű bájt van elöl, azaz az alacsonyabb memóriacímen. Angolul gyakran *little-endian*nak hívják, magyarul néha *alvég*nek, így van ez például az Intel gyártmányú processzorok esetében, ezért *Intel konvenció*nak is szokták nevezni.

**MSB** Most Significant Byte first

A legnagyobb helyiértékű bájt van elöl, azaz az alacsonyabb memória címen. Angolul gyakran *big-endian*nak hívják, magyarul néha *felvég*nek, így van (volt) ez például a Motorola gyártmányú processzorok esetében, ezért *Motorola konvenció*nak is szokták nevezni.

Mindkét sorrendet jelenleg is számos gyártó követi. A téma iránt mélyebben érdeklődőknek ajánljuk: https://en.wikipedia.org/wiki/Endianness

A témához kapcsolódóan felmerül egy másik érdekes kérdés is; ez a *bájtok bitsorrendje* (bit endianness, bit numbering), ami szintén lehet:

**lsb** least significant bit first

A legkisebb helyiértékű bit van elöl.

**msb** most significant bit first

A legnagyobb helyiértékű bit van elöl.

A bájt és bitsorrend jelzése között ezt a rendkívül logikus, nagy- és kisbetűs megkülönböztetést [8] vezette be, de sajnos a szakirodalom nem vette át, bitekre is inkább a nagybetűs jelölés használják.

Az FRC 1700 egyértelműen kimondja, hogy hálózati protokollok (RFC-kben való) dokumentációjában a big-endian sorrendet kell követni a több bájtból álló adatok esetén. Az átvitel során is balról jobbra kell haladni, azaz a legnagyobb helyiértékű bájtot kell először átvinni. Ezt *hálózati bájtsorrend*nek (network byte order) nevezik. Ezenkívül a számértékeket kifejező bájtok bitsorrendjéről is rendelkezik, ott is kötelezővé teszik az MSB sorrendet.

Következmény: mivel a hálózati bájtsorrend az MSB-t követi, így amennyiben az általunk használt architektúra nem ilyen, akkor hálózatot kezelő programok írása esetén ügyelnünk kell a konverzióra. Hordozhatónak szánt programok esetén úgy kell több bájtos adatokat kezelnünk, hogy mindkét fajta architektúrán helyesen működjön a programunk. (Ezenkívül természetesen az adatok átvitele esetén az egész számok eltérő hosszára is figyelnünk kell.)

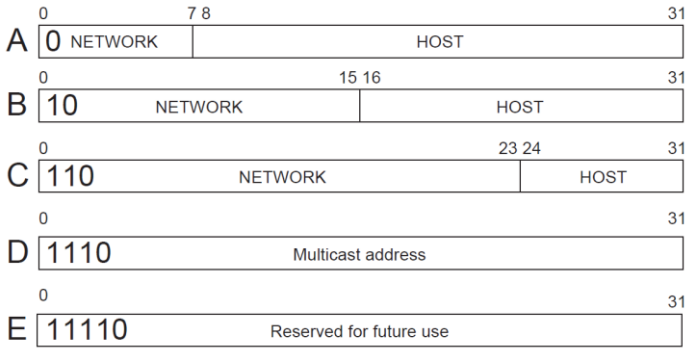
# Az Internet Protokoll 4-es verziója

Az IP protokollt eredetileg az RFC 791-ben definiálták.

## Az IP-címek

### Az IP-címek felépítése

Az IP-nek két, bárhol elhelyezkedő számítógép között képesnek kell lennie adatokat szállítania. Ehhez mindenek előtt megfelelő címzésre van szüksége. Amikor egy levelet postán szeretnénk elküldeni valakinek, akkor meg kell adnunk, hogy a címzett mely ország mely városában, ott milyen utcában és hányas szám alatt lakik. A címzés tehát hierarchikus. Az IP 32 bitet használ egy állomás megcímzésére. Az eredeti koncepció szerint a hierarchikus címzés két szinten valósult meg. Az IP-cím első része a *hálózatcím* (network address) kiválasztja azt a hálózatot, amelyikre az állomás csatlakozik, a második része a *gépcím* (host address) pedig a cím első része által kiválasztott hálózaton belül azonosítja a számítógépet. A protokoll tervezői úgy döntöttek, hogy több lehetőséget adnak arra, hogy hol legyen a két rész határa a 32 biten belül. Annak ismeretében alakították ki az *IP-cím osztályok*at, hogy nagy szervezetből kevés van, közepesből jóval több, kicsiből pedig kifejezetten sok. A cím első néhány bitje meghatározza, hogy az IP-cím melyik osztályba tartozik (A, B, C, D, E). A 3. ábra megmutatja, hogy melyik osztályban mekkora rész jut a hálózat, illetve a gép címének megadására. Az osztályokon alapuló *classful addressing* ma már részben csak történeti jelentőségű, a gyakorlatban már régóta *osztálymentes címzés*t (classless addressing) használunk.




*3. ábra. Az IP-címek felépítése (címosztályok alapján)*

Speciális jelentése van, ha egy IP-cím host részében minden bit 0 értékű: ez az egész hálózatot jelentő *network address*; hasonlóan annak is, ha a host részben minden bit 1 értékű: ez az adott IP hálózatba tartozó összes gépet jelentő *broadcast address*.

Az IP-címek *kanonikus* (egyezményes, szabványos) írásmódja a következő: A 32 bites címeket 4 db *oktett*re[6] bontjuk, majd ezeket tízes számrendszerben, egymástól ponttal elválasztva írjuk le. Például: 193.224.128.1. Ez a cím binárisan 110-val kezdődik, mert a 193 kettes számrendszerben: 11000001. Tehát ez a cím „C” osztályú. A hálózatcím 3 oktett, a gépcím 1 oktett.[7]

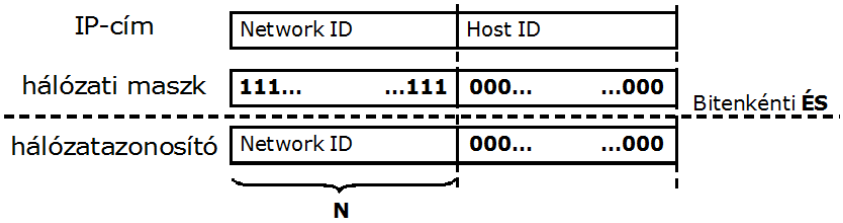
### Osztálymentes címzés

Bár az osztály alapú címzés annak idején logikus döntés volt, több probléma is volt vele. Az egyik az, hogy egy-egy szervezet hálózata több fizikai hálózatból épül fel, és ezért ezt az IP-címek kiosztásának is követnie kellett: alhálózatokat alakítottak ki (subnetting). Egy másik probléma volt az, hogy a B

[6] A bájt megnevezése a TCP/IP protokollcsalád fogalomtárában. Hangsúlyozottan 8 bitet jelent.

[7] Vegyük észre, hogy milyen kényelmes az, hogy az összes IP-cím osztályban a két rész határa mindig oktett határra esik!

osztályú címek nagyon gyorsan fogytak, míg a C osztályú tartományok túl kicsinek bizonyultak. Ezért több C osztályú tartomány összevonásával (supernetting) tudtak csak megfelelő méretű tartományokat biztosítani. A két megoldás mindegyikét magában foglalja az *osztálymentes címzés*, amikor az IP-címben a hálózatcím és a gépcím határát egy úgynevezett hálózati maszk (netmask) segítségével jelölik ki. Ez a maszk a hálózatcímnek megfelelő biteken csupa 1-est, a gépcímnek megfelelő biteken pedig csupa 0-t tartalmaz. Ha az IP-cím és a maszk között bitenkénti logikai ÉS műveletet végzünk, akkor megkapjuk a hálózatcímet (4. ábra). Vegyük észre, hogy ezzel a megoldással a két rész határát teljesen rugalmasan határozhatjuk meg! Mivel az 1-es bitek a maszk elején, a 0-s bitek pedig a maszk végén összefüggően vannak, ezért ha megadjuk, hogy hány darab 1-es van a maszkban, azzal a maszkot egyértelműen meghatároztuk. Amennyiben az egyesek számát az IP-cím után írjuk egy ferde vonallal („/”) elválasztva, akkor az alhálózatot egyértelműen, de rövidebb írásmóddal tudjuk megjelölni. Lássunk erre egy példát. Ahelyett, hogy: „IP-cím: 152.66.148.0, maszk: 255.255.252.0”, azt írhatjuk, hogy: 152.66.148.0/22.




*4. ábra. Hálózatcím előállítása hálózati maszk segítségével*

## Az IP hálózatcímek kiosztása

Az IP hálózatcímek kiosztását legfelső szinten az *Internet Assigned Number Authority* (IANA) végzi, amely /8 méretű blokkokat osztott ki az öt *Regional Internet Registry* (RIR) számára (amíg a készlete 2011-ben teljesen ki nem merült). Hazánk ezek közül a RIPE NCC-hez tartozik.

Vannak speciális célú tartományok, ezek közül fontos megemlíteni a *privát*, más szóval *nem publikus* IP-cím tartományokat, amelyeket bárki használhat, anélkül, hogy igényelné, de ezek használatával csak a saját intézményén belül tud kommunikálni, a világ többi része felé nem. (A címfordítástól[8] eltekintve, de ahhoz is kell egy darab publikus IP-cím.) Ezekből az érdeklődő hallgatóink bátran oszthatnak a saját otthoni számítógépeik számára IP-címeket, ha hálózatba szeretnék kötni őket.

Az alábbiakban röviden összefoglaljuk a legfontosabb speciális célú IPv4 címeket:

- **Host ID: csupa 0** – Hálózat címe – az adott hálózat eszközei (elvileg) opcionálisan kezelhetik, de nem szokták.
- **Host ID: csupa 1** – Broadcast cím – (directed broadcast; elavult kifejezéssel: subnet broadcast), az adott hálózaton mindenkinek szól.
- **127.0.0.0/8** – Loopback cím – A helyi gépet azonosítja, bármelyik használható, a 127.0.0.1 a szokásos.
- **Privát IP-címtartományok (RFC 1918)** – Csak helyi hálózaton (Interneten nem) érvényes címek – ilyenek: 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16.
- **Link lokális IP-címtartomány** (RFC 3927) – 169.254.0.0/16 – Csak helyi kommunikációra, a routerek nem továbbítják! Ebből az automatikus címkonfigurációhoz használható címekbe az első és utolsó 256 db cím nem tartozik bele, ezeket további célokra tartják fenn!

[8] NAT: Network Address Translation – e jegyzet keretében bővebben nem tudunk vele foglalkozni.

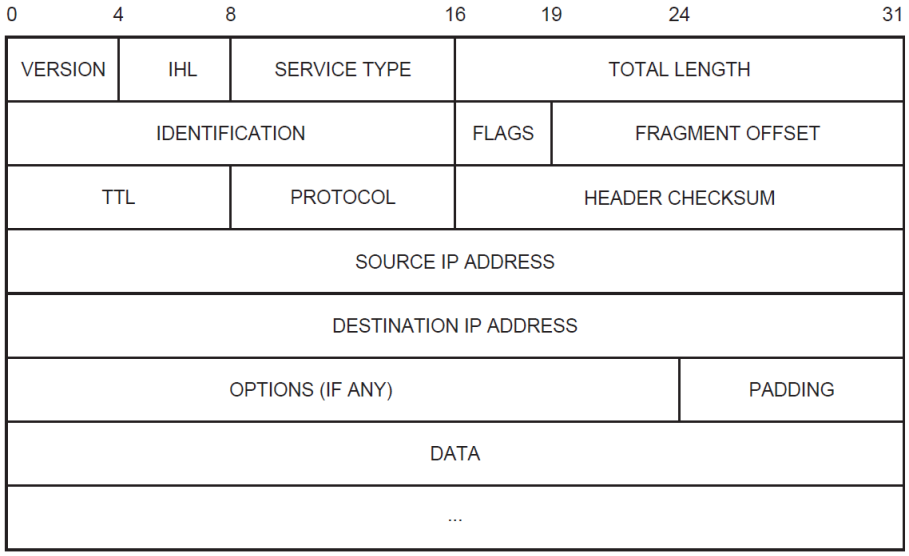
- **Dokumentációs célokra lefoglalt IP-címtartományok** (RFC 5737) – 192.0.2.0/24 (TEST-NET-1), 198.51.100.0/24 (TEST-NET -2), 203.0.113.0/24 (TEST-NET -3) Ezek célja, hogy ha példákban használják őket, és valaki meggondolatlanul begépeli, akkor ne okozzon ütközést valóságos rendszerekkel. (Ettől még ezeket nem szabad használni.)

*További speciális célra lefoglalt tartományok: RFC 6890*

Megjegyzés: A „D” osztályú címek (224.0.0.0/4) az eredeti koncepciónak megfelelően ma is multicast céljára szolgálnak, míg az „E” osztályú címek (240.0.0.0/4) gyakorlatilag elvesztek (mivel bizonyos hálózati eszközök nem továbbítják őket, ezért végül nem is osztották ki őket).

## Az IP datagramok felépítése és használata

Az IP adategységét *datagram*nak nevezik. Az IP datagram felépítését az 5. ábra mutatja be, melyen minden lehetséges mezőt feltüntettünk.




*5. ábra. IP datagram felépítése*

Az egyes mezők nevét rövidítettük, ezeknek megadjuk a teljes nevét, valamint megadjuk az egyes mezők funkcióját is, közben lépésről lépésre megismerkedünk az internet protokoll működésével.

**Version** verziószám

Értéke: 0100, mivel az IP 4-es verziójáról van szó. (A későbbiekben megismerendő IPv6 esetén értéke: 0110, de a megkülönböztetés már link szinten megtörténik.)

**IHL** (Internet Header Length) fejrész hossza

1 egység = 32 bit (4 oktett), ezért a fejrész oktettekben mért hosszának oszthatónak kell lennie 4-gyel. Tipikus értéke az 5; ha vannak opciók a fejrész végén, akkor lehet 6 vagy több.

**Type of Service** szolgálat típusa

Ez a mező további almezőkre bomlik, ráadásul az értelmezése változott, ezért külön foglalkozunk vele.

**Total Length** a datagram teljes hossza

**Identification** azonosítás

Szétdarabolt keretek esetén azonosítja az összetartozó töredékeket.

**Flags** jelzőbitek

Három bitből áll, ezek rendre:

**0** (reserved) kihasználatlan

Értéke kötelezően 0.

**DF** (Do not Fragment) ne tördeld

0: szabad tördelni

1: nem szabad tördelni

**MF** (More Fragments) van még töredék

0: ez az utolsó töredék

1: ez még nem az utolsó töredék

**Fragment Offset** töredék eltolás

Tördelés esetén megadja, hogy az adott töredék adatrészében a kezdő bájt hányadik volt az eredeti datagram adatrészében. Mivel a mérete 13 bit, ezért 8 oktettes egységekben értendő! (A datagramok tördelésének leírása megtalálható [1] jegyzetben.)

**Time to Live** élettartam

Legfeljebb 255 lehet az értéke, minden csomópontnál csökkentik az értékét a várakozással eltöltött másodpercek számával, de legalább 1-gyel. (A gyakorlatban 1-gyel.) Ha 0-ra csökken az értéke, a csomagot eldobják. Ennek a célja az, hogy ha esetleg az útválasztás hibája miatt egy csomag „eltéved”, ne maradjon korlátlanul hosszú ideig a hálózatban (erőforrás pazarlás).

**Protocol** protokoll

Megadja, hogy az IP felett milyen protokoll helyezkedik el. (pl. TCP, UDP, ICMP)

**Header Checksum** a fejrész ellenőrző összege

Csak a fejrész mezőire képezik. Amennyiben a vett datagram ellenőrző összege hibás, akkor a datagramot eldobják.

**Source Address** forrás IP-címe

**Destination Address** cél IP-címe

**Options** opciók

Szerepelhetnek egyéb beállítások is, de nem feltétlenül vannak. Ha vannak, a méretük nem feltétlenül a 4 oktett egész számú többszöröse.

**Padding** helykitöltés

Mérete 1, 2 vagy 3 oktett lehet. Azért szükséges, mert az „IHL” egysége 32 bit, így a fejrész méretét mindig ki kell egészíteni annak többszörösére.

**Data** adatok

Itt található beágyazva az IP fölötti protokoll adategysége.

### A Type of Service mező régi és új értelmezése

Már az IP protokoll tervezésekor is gondoltak arra, hogy attól függően, hogy mit szállít egy datagram, más-más elbánásra lehet szüksége. A Type of Service mezőt arra tervezték, hogy ezt támogassa. Az eredeti, RFC 791-ben leírt mezőit látjuk a 6. ábra bal oldalán.


*6. ábra. A Type of Service mező régi és új felépítése*

Ezek értelmezése a következő:

- Az első három bit a prioritás (elsőbbség) megadására szolgál.
- A D, T és R bitek közül legfeljebb két darab értéke lehet 1-es. Amelyik értéke 1-es, az útválasztók a datagram továbbítása során annak a jellemzőnek az értékét igyekeznek optimalizálni, de garanciát nem vállalnak. Úgy nevezik ezt, hogy a hálózat „best effort” jellegű.

  **D** (Delay) 1-es értéke azt jelenti, hogy a késleltetés minél kisebb értéke a fontos számunkra.

  **T** (Throughput) az időegységenkénti minél több adat átvitelét jelenti.

  **R** (Reliability) pedig a megbízhatóságot, hibamentességet kéri.
- Az utolsó két bitet eredetileg nem használták, értékük kötelezően 0.

Azonban a routerek általában nem vették (és legtöbbször ma sem veszik) figyelembe a Type of Service mezőben található értékeket. A Type of Service mező használatának megváltoztatására többféle javaslat is született, ezek közül ma is érvényesek (de vannak további kiegészítéseik is):

**RFC 2474** Használjuk a 8 bitből a baloldali 6 bitet (RFC 791 szerinti prioritás és DTR) *különleges kiszolgálás* (Differentiated Services, szó szerint: megkülönböztetett szolgáltatások) nyújtása érdekében. Ezt a 6 bitet DSCP-nek (Differentiated Services Code Point) nevezik.

**RFC 3168** Használjuk az utolsó két bitet Explicit Congestion Notification (ECN) célokra.

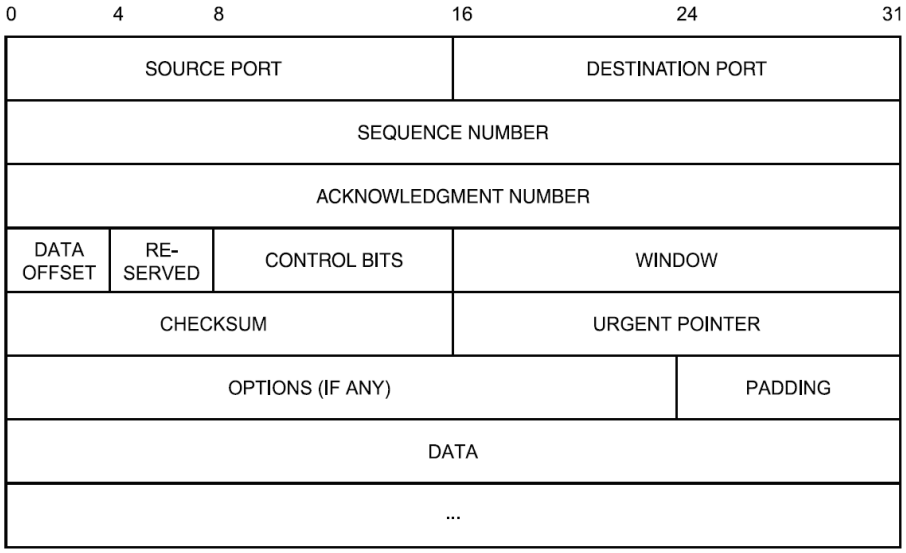
Tehát a jelenlegi állás szerint a Type of Service mező első 6 bitje DSCP, az utolsó két bitje pedig ECN célokra szolgál (6. ábra jobb oldala). Ezekkel e segédlet keretében mélyebben nem foglalkozunk.

# Transmission Control Protocol

A *Transmission Control Protocol*t az RFC 793-ban definiálták. A TCP megoldja a megbízható átvitelt is, és a kapcsolatok azonosítását is. Egy kapcsolat két végpontja között megbízható átvitelt biztosít kétirányú adatforgalommal: ami az egyik végponton elindul, a másikon pontosan annak, és ugyanolyan sorrendben kell megjelennie. A hálózati kapcsolat *végpontját* pedig a *portszám* azonosítja. A portok mindig konkrét szolgáltatást azonosítanak. Például a 80-as porton webkiszolgáló, a 22-es porton ssh szerver működik, stb. Egy *TCP kapcsolatot* pedig 4 szám azonosít egyértelműen: a küldő és a címzett gép IP-címe, valamint a küldő és a címzett gépen a portszámok. (Így ha például ugyanazon két gép között van két azonos szolgáltatást igénybe vevő kapcsolat, akkor a négyből három szám (a két IP-cím és a szerver portszáma) megegyezik, de a negyedik alapján akkor is megkülönböztethető a két TCP kapcsolat.)

## A TCP szegmens felépítése

A TCP adategységét *szegmens*nek hívják. (Egy TCP szegmens mindig egy IP datagram adat mezőjében utazik.) A TCP szegmens felépítését a 7. ábra mutatja be.




*7. ábra. TCP szegmens felépítése*

Egy TCP szegmens az alábbi mezőket tartalmazza:

**Source Port** forrás port

A portszám azon a gépen, ahonnan a szegmenst küldték.

**Destination Port** cél port

A portszám azon a gépen, ahova a szegmenst küldték.

**Sequence Number** sorszám

Az átvitel során az oktetteket sorszámozzuk. A sorszám megmutatja, hogy a szegmens adatmezőjének kezdő oktettje hányas sorszámot kapott.

**Acknowledgment Number** nyugta

Annak az oktettnek a sorszámát adja meg, amelyiket várja; addig az összes nyugtázva van.

**Data Offset** adatmező eltolás

32 bites egységekben mérve megadja az adatmező kezdetét a TCP szegmens kezdetéhez képest. Tulajdonképpen a fejrész hossza, mint IP-nél az IHL mező.

**Reserved** későbbi használatra fenntartva

Az RFC 793 szerint még a 4 bites Data Offset mező után következő 6 bit nem volt használatban. Ez jobbról 2-vel csökkent RFC 3168 alapján.

**Control bits** vezérlőbitek

RFC 793 szerint csak 6 darab volt, az RFC 3168 vezette be a CWR és az ECE vezérlőbiteket a korábban fenntartott 6 bites terület végén.

A vezérlőbiteknél mindig azok 1 értéke esetén áll fenn az, amit a nevük jelent.

**CWR** (Congestion Window Reduced) – RFC 3168

A torlódási ablakot szűkítették. Bővebben nem foglalkozunk vele.

**ECE** (ECN-Echo) – RFC 3168

Torlódásjelző visszhang. Bővebben ezzel sem foglalkozunk.

**URG** (urgent)

A *sürgős adat mutató* érvényes.

**ACK** (acknowledgment)

A *nyugta* mező érvényes.

**PSH** (push)

Jelzi az adat késedelem nélküli továbbításának igényét.

**RST** (reset)

Egy host összeomlása, vagy más okból összezavart összeköttetés helyreállítására szolgál, illetve ha egy porton semmilyen program sem figyel, akkor az adott portra megkísérelt kapcsolatfelépítés esetén mindjárt az első SYN csomagra RST a válasz.

**SYN** (synchronize)

Összeköttetés létesítésére szolgál.

**FIN** (finish)

Összeköttetés bontására szolgál.

**Window** ablak

Az ablakméretet adja meg. A megbízható átvitelhez szükséges nyugtázási mechanizmus használja, részletesen foglalkozunk vele.

**Checksum** ellenőrző összeg

A teljes szegmensre képezik, sőt még egy 12 bájtos ún. *pseudo header*t is tesznek a szegmens elé, ami többek között a forrás és cél IP-címeket is tartalmazza, az adategység végén pedig esetleg egy 0 értékű oktettel kiegészítik, ha anélkül páratlan oktettből állna (mert 16 biten történik az ellenőrző összeg kiszámítása). Amennyiben a vett szegmensben az ellenőrző összeg hibás, akkor azt eldobják.

**Urgent Pointer** sürgős adat mutató

A *sürgős adat* az adatfolyam menetét megszakítva a többi adatot megelőzve feldolgozandó adat. A sürgős adat az adatmező elejétől kezdődik, a mutató a sürgős adat után következő (már nem sürgős) adat kezdetére mutat.

**Options** opciók

Mint IP-nél.

**Padding** helykitöltés

Mint IP-nél. (A magyarázatban természetesen IHL helyett Data Offset értendő.)

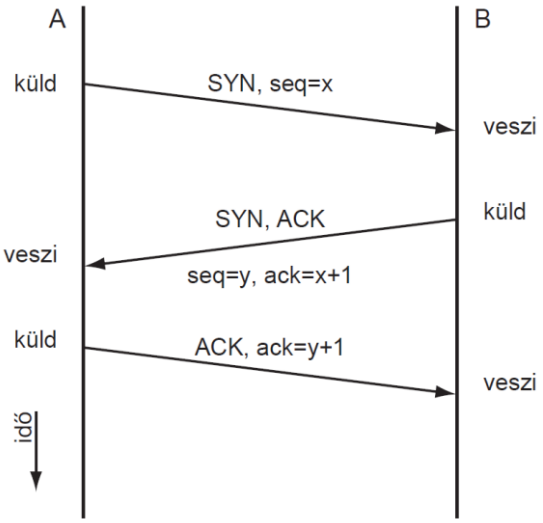
**data** adatok

Itt található beágyazva a TCP fölötti protokoll adategysége.

Az IP-vel szemben a TCP kapcsolatorientált protokoll, így nem véletlen, hogy a mezők jelentős részének a használata nem magyarázható meg kielégítően egy-két mondattal. Most egyenként megvizsgáljuk, hogyan történik a *kapcsolat felépítése*, a *megbízható átvitel*, a *forgalomszabályozás*, a *torlódásvezérlés* és végül a *kapcsolat lebontása*.

## TCP kapcsolat felépítése

Tegyük fel, hogy egy „A” állomás kapcsolatot szeretne kezdeményezni egy „B” állomással. Az „A” állomás lefoglal egy *kapcsolatleíró*t, amiben a kapcsolat paramétereit tárolja, majd összeállít egy TCP szegmenst a „B” számára: generál egy *kezdő sorszámot*[9] (*x*), amit elhelyez a *sequence number* mezőben, valamint beállítja a SYN vezérlőbit értékét 1-re; ezzel jelzi, hogy ez egy új kapcsolat. A szegmenst (az IP réteg segítségével) elküldi „B”-nek. Kövessük ezt nyomon a 8. ábrán! A „SYN” feltüntetése azt jelenti, hogy ennek a vezérlőbitnek az értéke 1, a *seq=x* önmagáért beszél.




*8. ábra. TCP kapcsolat felépítése*

Amikor a „B” állomáshoz megérkezik a kapcsolat kezdeményezése, a „B” állomás is lefoglal egy kapcsolatleírót, amiben eltárolja az *x* értéket (és a továbbiakban ebben tartja nyilván a kapcsolat paramétereit). A „B” állomás összeállít egy válasz szegmenst. Ehhez „B” is generál egy kezdő sorszámot (*y*), amit elhelyez a válasz szegmens *sequence number* mezőjében, és beállítja a SYN vezérlőbitet. Ezen kívül a szegmens *acknowledgment number* mezőjébe az *x*+1 értéket helyezi el[10] és beállítja az ACK vezérlőbitet. Ezután elküldi a szegmenst „A”-nak. Összefoglalva: „B” létrehozta az „A” fele irányuló kapcsolatra vonatkozó sorszámot (*y*) és nyugtázta az „A” által küldött sorszámot. Kövessük az ábrán!

[9] A modern operációs rendszerek (például OpenBSD) kriptográfiailag erős véletlenszám generátort használnak, mert ha bármilyen módon előrejelezhető lenne a kezdő sorszám, az támadásra adna lehetőséget.

[10] Ezzel tulajdonképpen azt állítja, hogy „A”-tól az adatokat *x* sorszámmal bezárólag vette, *x*+1-től várja.

Végül „A” állomás veszi a „B” állomástól érkező TCP szegmenst, és eltárolja a kapcsolatleírójában a „B”-től vett *y* sorszámot. Létrehoz egy harmadik TCP szegmenst, a nyugta mezőbe beírja az *y*+1 értéket, és beállítja az ACK vezérlőbit értékét. Az így elkészített szegmenst elküldi „B”-nek, aki veszi azt. Ezzel a kapcsolat mindkét irányban felépült, a résztvevő állomások mindkét irányú adatfolyamra vonatkozólag egyeztették a sorszámokat, és a nyugtázás is megtörtént. Kezdődhet az adatforgalom.

Ezt a módszert *háromutas kézfogás*nak (three way handshake) nevezzük.

## A megbízható átvitel

Az IP nem megbízható átvitelére építve a TCP megbízható átvitelt nyújt mindkét irányban. Ehhez mindkét irányban sorszámozza az adatfolyam oktettjeit, valamint ellenőrző összeget használ. Amikor megérkezik egy szegmens, a TCP protocol stack kiszámítja a szegmens ellenőrző összegét, és összeveti a benne szereplővel. Ha nem egyezik, akkor nem használja fel a benne szereplő adatokat. Ha az ellenőrző összeg helyes, akkor még meg kell vizsgálnia a sorszámot (*sequence number*), hogy vajon egyezik-e azzal, amit várt. Ha a vártnál kisebb a sorszám, akkor feltehetően valami „kóbor” szegmensről van szó (például korábban nem érkezett meg időben, újra kérték és most megjött, vagy esetleg a hordozó hálózat hibájából egy adategység megkettőződött), amit nem fogad el. Ha a sorszám a vártnál nagyobb, akkor sem fogadja el.[11] Tehát csak akkor fogadja el, ha a sorszám pontosan egyezik a várt értékkel. Ekkor *előbb vagy utóbb* nyugtát küld a megfelelő értékkel. A megfelelő érték kiszámítása nagyon egyszerű: ha *x* a sorszám és *z* oktett érkezett, akkor a nyugta értéke: *x*+*z*. De mikor is kell a nyugtát elküldeni? A hálózat hatékony kihasználása érdekében nem azonnal. Amennyiben van mit küldeni az ellenirányú adatfolyamban, akkor a nyugtát mintegy „ráültetik” arra az adatfolyamra, vagyis a másik irányú adatfolyam TCP szegmensének a nyugta mezőjét használják fel. Ezt angolul úgy hívják, hogy *piggybacking*, magyarul *ráültetett nyugta* a neve. Természetesen abban az esetben, ha egy meghatározott ideig nincs küldeni való az ellenirányú adatfolyamban, akkor elküldünk egy TCP szegmenst kizárólag a nyugtázás céljából, hiszen a küldő félnek adott (a kapcsolat során megfelelő algoritmussal dinamikusan változtatott) idő (timeout) eltelte után újra kell küldenie az adatot, ha addig nem kapott nyugtát.

## A forgalomszabályozás

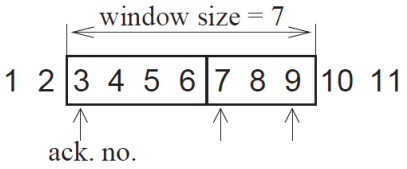
Egyáltalán miért van szükség *forgalomszabályozás*ra (flow control)? Ennek megvilágítására tekintsük a következő szituációt. Egy „A” állomás (mint forrás) nagy mennyiségű adatot szeretne küldeni egy tőle távoli „B” állomásnak (mint célnak). A kommunikáció sebessége függ mindkét állomás és a közöttük levő hálózat sebességétől. Most koncentráljunk csak a két gépre (a hálózati *torlódás* kezelését külön tárgyaljuk). Amennyiben „A” minden újabb adategység küldése előtt megvárná az előzőre adott nyugtát, az nagymértékben lassíthatná a kommunikációt. Nézzünk erre egy számpéldát: legyen „A” és „B” távolsága 200 km, az őket összekötő üvegszál törésmutatója *n*=1,5, a fénysebesség 300000 km/s (vagyis az üvegben 200 km/ms), az átviteli sebesség 100 Mbit/s. Ha egyszerre 1000 bájt hasznos adatot küldünk, és a beágyazás során (bőkezűen számítva) még 100 bájt adódik hozzá, akkor a 8800 bit leadásához 88 µs szükséges. Az oda-vissza út ideje csak a terjedési időt számítva is 2 ms. Ez azt jelenti, hogy hiába lenne szabad az átviteli csatorna, még az idő (és így a csatornakapacitás) 1/20 részét sem tudjuk kihasználni! És ennél jóval nagyobb távolságok és átviteli sebességek is léteznek, ahol az arány sokkal rosszabb! Tehát egy állomás nem várhat az elküldött adategységének a nyugtájára, mielőtt újabbat elküldene. Akkor tehát nem fog várni. Így azonban egy másik probléma merül fel: amennyiben „A” lényegesen gyorsabban küldi az adatokat (és a hálózat képes átvinni), mint ahogyan „B” képes azt feldolgozni, előbb-utóbb megtelnek „B” pufferei és az

[11] Az alap koncepció szerint. Létezik olyan megoldás is, ahol elfogadja, lásd: *TCP Selective Acknowledgment Options*, RFC 2018.

adatok elvesznek. Ekkor a „B”-nél elvesző forgalom a hálózat kapacitását fölöslegesen használja. Márpedig léteznek erősen különböző sebességű számítógépeink, tehát a probléma valós. Szükség van tehát egy megoldásra! Ha „B” jelezni tudná, hogy vette ugyan „A” adását, de egy ideig többet nem tud fogadni, az még nem jelentene megoldást, hiszen mire a jelzés visszaér „A”-hoz, arra sok adat elindul feleslegesen. Jobb megoldás kell!

A TCP a *csúszó ablak* (sliding window) módszert használja forgalomszabályozásra. Az ablak értéke egy hitelkeretnek tekinthető, azt fejezi ki, hogy egy állomás ennyi adat elküldését engedélyezi a másik fél számára úgy, hogy a másik fél még nem kapott nyugtát róla.

A kapcsolat felépülésekor az állomások megegyeznek a kezdő ablakméretben is, amit később (látni fogjuk milyen módon és feltétellel) meg is változtathatnak. A módszer működésének megértése érdekében nézzünk meg egy számpéldát kis számokkal!




*9. ábra. A csúszóablak működése*

A 9. ábrán az ablak bal széle az utolsó nyugta értékétől kezdődik (ez 3, ami azt jelenti, hogy 2-ig nyugtázva, a 3 következik) az ablak mérete pedig 7. Ez a küldő számára azt jelenti, hogy a 3-tól 9-ig terjedő sorszámú, összesen 7 darab oktett az, amit anélkül elküldhet, hogy nyugtát kapna. Például elküldi a 3-6 sorszámú oktetteket. Ekkor nem kell nyugtára várnia, hanem küldheti a 7-9 sorszámúakat is. Hogyan csökkentheti a fogadó fél az ablakméretet? Esetünkben a 3-6 sorszámú oktettek vétele után nyugtázza őket (nyugta értéke: 7) és az ablakméretnek legalább 3-nak kell lennie! Ugyanis ha csak 2 lenne, akkor a 9-es ezután nem lenne küldhető, pedig korábban az volt. Legyen most az ablakméret 5, ekkor a 7-11 sorszámú oktettek küldhetők. Az ablak tehát a folyamat során csúszik előre, ezért hívják *csúszó ablak*nak.

A bevezető számpélda után felmerül a kérdés, hogy vajon a TCP-ben használható legnagyobb ablakméret elég nagy-e? A 16 bites mező 64 kB-os[12] ablakméretet tesz lehetővé, ami 512 kbit. Ennek az átvitele (a fejrészeket elhanyagolva) a példában említett 100 Mbit/s-os hálózaton 5,12 ms-ig tart, ami nagyobb a 2 ms-nál, viszont ugyanabban a nagyságrendben van. Márpedig léteznek gigabites hálózatok, és a 2000 km sem tekinthető a valóságtól elrugaszkodottnak. Erre bizony azt kell mondanunk, hogy az eredeti TCP-vel ebben az esetben nem tudjuk egyetlen kapcsolattal kihasználni a hálózat kapacitását. Ez a mindennapi élet szempontjából azért nem okoz súlyos problémát, mert az ilyen nagy távolságú és sebességű hálózatok tipikusan gerinchálózatok, amit nem egy felhasználó forgalmával kell kitöltenünk. Azt azonban látnunk kell, hogy a csúcstechnológiát illetően az eredeti TCP tartalékai kimerültek. Természetesen van megoldás a problémára: a TCP *Window Scale* opció használata (RFC 7323). Ennek lényege, hogy a TCP kapcsolat felépítésekor a SYN szegmensben (amikor a felek a kezdő sorszámot és ablakméretet is közlik egymással) megadnak egy *n* számot, ami azt jelenti, hogy a későbbiekben az ablakméret (Window) mező bináris értékét a másik fél használat előtt *n*-szer tolja el balra, ami $2^n$-nel való szorzást jelent. (Például n=8 esetén a 8 balra tolás $2^8$=256-tal való szorzásnak felel meg.) A megoldás előnye, hogy az opciót csak a kapcsolat felépítésekor kell használni, a későbbiekben nincs szükség rá. A megoldás következménye, hogy a ténylegesen használt ablakméret mindig $2^n$ többszöröse, de ez nem okoz problémát.

[12] Egészen pontosan ennél egy bájttal kisebbet, de most ez nem számít.

## Torlódásvezérlés

A *torlódásvezérlés* (congestion control) célja annak az elkerülése, hogy valamely közbenső hálózati eszköz (router, link) túlterhelése miatt a hálózat teljesítőképessége radikálisan csökkenjen (congestive collapse).

Miért is fordulhatna elő ilyen állapot?

- Ha a hálózat terhelése túl nagy (megközelíti a kapacitását), akkor a csomagvesztés megnő, és a TCP elkezdi újra küldeni a nyugtázatlan szegmenseket.
- Az újraküldés okozta terhelésnövekedés további csomagvesztéseket okoz, és az önmagát gerjesztő folyamat odáig jut, hogy a hálózat kapacitásának csak a töredékét képes átvinni rendkívül rossz minőségi jellemzők mellett.

A TCP torlódásvezérlésre számos algoritmus létezik, az aktuális szabvány kereteit az RFC 5681 adja meg. A torlódásvezérlésre használt algoritmusok bevezetik a *torlódási ablak* (congestion window) fogalmát. Figyelem! Ez nem azonos a TCP *ablakméret* (Window) vagy más néven *vételi ablak* (receive window) paraméterével! A torlódási ablak célja, hogy korlátozza két kommunikáló fél között a nyugtázatlan szegmensek (pontosabban a bennük levő adat oktettek) számát. De a TCP ablakméret paraméterével szemben ennek mérete nem a két kommunikáló fél képességeitől, hanem a hálózat aktuális áteresztő képességétől függ. Az algoritmusok a torlódási ablak méretét attól függően változtatják, hogy tapasztalnak-e torlódásra utaló jeleket; ha ilyen jelek nincsenek, a méretét növelik, ha vannak, akkor csökkentik. Természetesen egy host adásakor a nyugtázatlanul elküldhető adatmennyiséget mindkét ablak mérete korlátozza:

küldhető oktettek száma = min(a vevő által megadott vételi ablak, torlódási ablak)

Megjegyzés: a torlódási ablak értékét a kommunikáló felek helyben tarják nyilván, egymással nem közlik, így külső megfigyelő ezt nem láthatja.

Mik lehetnek a *torlódásra utaló jelek*?

- Egy szegmens küldése után a (hálózati viszonyok alapján adaptívan állított) timeout letelt és nem jött nyugta. Ennek az oka hálózati torlódás is lehet, de persze más is (például bithiba miatt egy IP router eldobta a szegmenst vagy annak nyugtáját szállító datagramot).
- Egy szegmensre többszörös nyugta érkezett. Ennek több oka lehet, például:
    - Torlódás miatt egy szegmens vagy annak nyugtája késett, emiatt a szegmenst újra elküldték. A nyugta aztán mindkét szegmensre megjött. – Ebben az esetben valóban torlódásra utal!
    - Csomagok sorrendje felcserélődött: egy később küldött csomag előbb érkezett meg, és a fogadó fél ezzel jelzi, hogy nem azt várta, hanem egy másik, korábban küldött csomagot (kisebb sorszámmal). – Ebben az esetben nem biztos, hogy a sorrendcsere oka torlódás!
    - A nyugtát szállító IP datagram megduplázódott az IP alatti réteg hibája miatt. – Ez egyáltalán nem jelent torlódást.
- Az RFC 3168 szerinti *Explicit Congestion Notification* érkezett.

Még két felhasznált fogalom:

- RTT: Round-Trip Time: az az időtartam, ami TCP szegmens elküldésétől a szegmens nyugtázására alkalmas nyugta megérkezéséig eltelik. (Arra gondolunk, hogy a nyugtát a szegmens váltotta ki, de lehet, hogy valójában nincs ok-okozati összefüggés köztük, csak a kapott nyugta nem megkülönböztethető.)
- MSS: Maximum Segment Size: a TCP szegmens számára megengedett maximális méret. Ez az IP-t szállító hálózat MTU-jától függ, a TCP kapcsolat felépítésekor a felek TCP opcióval megadhatják egymásnak ezt a paraméterüket.

Az egyik legfontosabb megoldás magja az *AIMD – Additive Increase / Multiplicative Decrease* algoritmus. Az algoritmus lényege, hogy a növelés fix értékek hozzáadásával, a csökkentés viszont 1-nél kisebb számmal való szorzással történik. Az algoritmus általánosabb, mint nekünk kell, a pontos leírása megtalálható: http://en.wikipedia.org/wiki/Additive_increase/multiplicative_decrease.

A TCP-nél használt változatát az RFC 5681-ben *torlódáselkerülés* (congestion avoidance) névvel illetik, és működése a következő:

Minden RTT alatt:

- Növeljük a torlódási ablak értékét MSS-sel, ha nincs torlódásra utaló jel.
- Csökkentsük a torlódási ablak értékét a felére, ha van torlódásra utaló jel.

Vegyük észre, hogy a növelés csak lineáris (azaz viszonylag lassú), a csökkentés viszont exponenciális; torlódás esetén nagyon gyorsan a felére, negyedére, stb. tud csökkenni.

Kezdőértékként használható egy fix érték, jobb híján akár MSS is. Azonban az algoritmust megelőzően az ún. *lassú kezdés* (slow start) algoritmust szokták használni, aminek csak egyik jellemzője a lassú kezdés, a másik éppen a gyors (exponenciális) növekedés.

A slow start működése:

Kezdetben torlódási ablak = MSS. (Pontosabban MSS méretétől függően kb. 2xMSS vagy 3xMSS, de ez részletkérdés.) Minden RTT alatt:

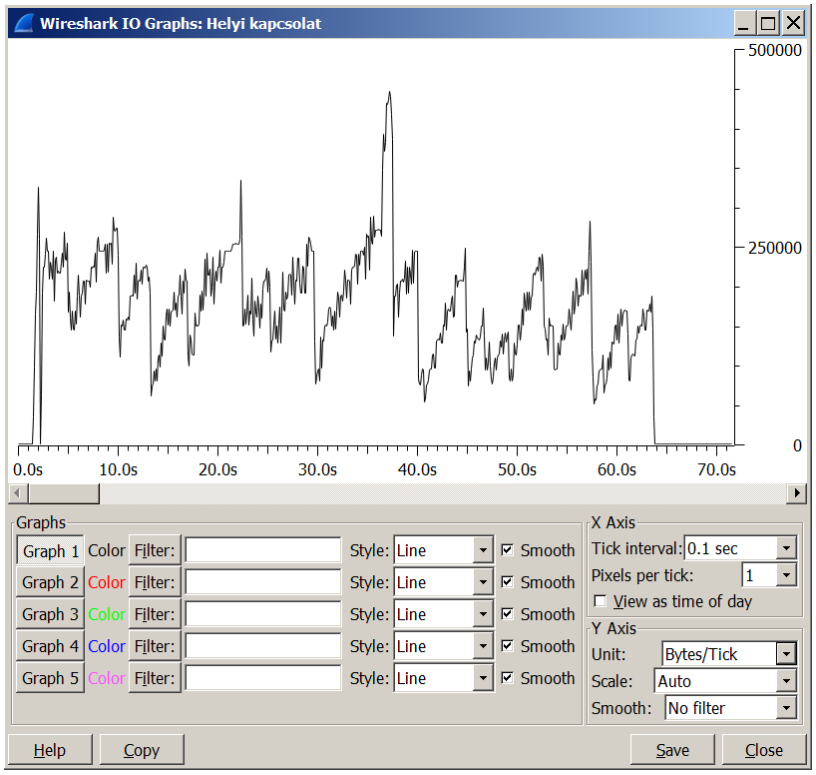
- Növeljük a torlódási ablak értékét a duplájára, ha nincs torlódásra utaló jel.
- Fejezzük be az algoritmust, ha van torlódásra utaló jel, vagy elértünk egy küszöbértéket (Slow-Start Threshold).

A slow start befejezésekor áttérünk a torlódáselkerülés algoritmusra.

A kettő kombinációjának az előnye, hogy a slow start segítségével kezdetben kis értékről gyorsan növekszik a torlódási ablak, de amint torlódásra utaló jel van, rögtön átvált a stabil torlódáselkerülés algoritmusra.

Megjegyzés: Az RFC 5681 illetve leír még két további algoritmust is (fast retransmit, fast recovery). Ezekkel mélyebben nem foglalkozunk.

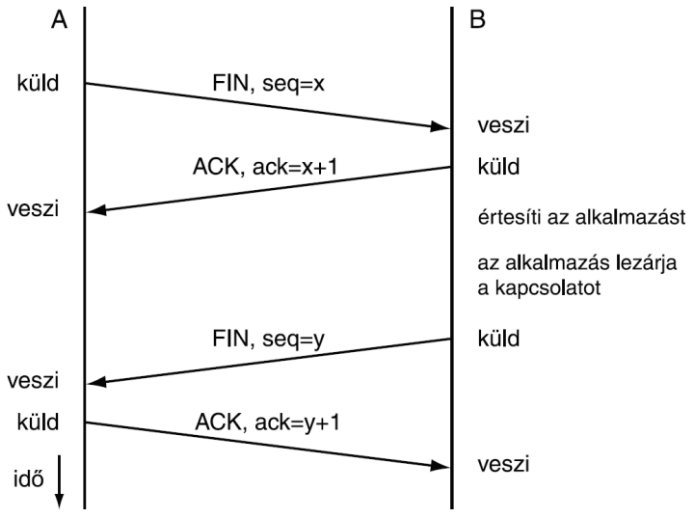
Amint korábban említettük, a torlódásvezérlés működése a hálózaton áthaladó adategységek vizsgálatával közvetlenül nem figyelhető meg, de az időegységenként átvitt adatmennyiség változása alapján igen. A 10. ábra egy ilyen vizsgálat eredményét mutatja be. Jól megfigyelhető rajta az AIMD algoritmus működése: a másodpercenként átvitt bájtok száma (througput) általában lineáris jelleggel nő, és időnként hirtelen esik vissza.




*10. ábra. A torlódásvezérlés működésének megfigyelése Wireshark segítségével*

## TCP kapcsolat lebontása

A TCP kapcsolat lebontását a 11. ábra mutatja be. Ha az „A” állomásnak nincs több küldeni valója, a FIN vezérlőbit beállításával jelzi a „B” állomás számára, hogy kéri a kapcsolat bontását. A „B” állomás a beállított FIN vezérlőbit alapján értesíti az alkalmazást, hogy „A” a kapcsolat bontását kérte, valamint nyugtázza az „A”-tól származó sorszámig az adatok vételét. Amikor a „B” állomáson futó alkalmazás is úgy dönt, hogy lezárja a kapcsolatot, beállítja a FIN vezérlőbitet az „A” felé küldött TCP szegmensben. Az "A" állomás ezt veszi, nyugtát küld, és bontja a kapcsolatot. A nyugta megérkezésekor a „B” is bontja a kapcsolatot. Ezt a megoldást nevezzük *négyutas kézfogás*nak (four way handshake).

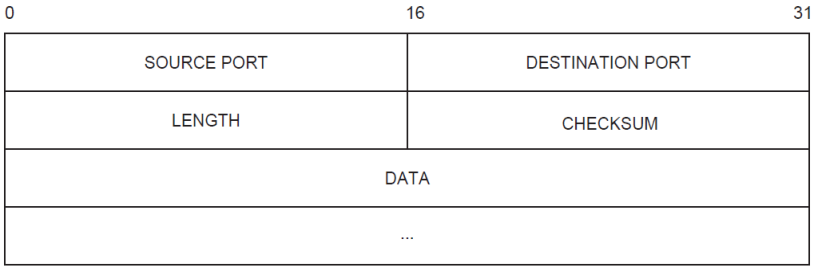



*11. ábra. TCP kapcsolat lebontása*

## User Datagram Protocol

A *User Datagram Protocol*t az RFC 768-ban definiálták. Vannak olyan esetek, amikor nincs szükségünk megbízható összeköttetésre, de az alkalmazások közötti kommunikációban akkor is szükség van a portokra az alkalmazások azonosításához. A User Datagram Protocol (UDP) kapcsolatmentes szállítási protokoll: datagramok küldését teszi lehetővé a felhasználók számára összeköttetés létesítése nélkül. Egy olyan kliens-szerver alkalmazás (például DNS névfeloldás) számára, amely egyetlen rövid kérésre egyetlen rövid választ küld, sokkal előnyösebb az UDP használata, mint az, hogy TCP kapcsolat felépítésével és lebontásával töltse az időt. Ezen kívül a valós idejű forgalom számára is alkalmatlan a TCP az újraküldési mechanizmusa miatt.

Az UDP adategységét *user datagram*nak hívják, felépítése a 12. ábrán látható.




*12. ábra. Az UDP adategységének felépítése*

Egy user datagram az alábbi mezőket tartalmazza:

**Source Port** forrás port

A portszám azon a gépen, ahonnan a datagramot küldték.

**Destination Port** cél port

A portszám azon a gépen, ahova a datagramot küldték.

**Length** hossz

A user datagram teljes méretét adja meg.

**Checksum** ellenőrző összeg

A TCP-hez teljesen hasonlóan történik a kiszámítása, itt is használják a *pseudo header*t. Elétérés viszont, hogy az ellenőrző összeg használata IPv4 esetén nem kötelező. (Ha nem használják, akkor azt a mező 0 értékével jelzik.) IPv6 esetén kötelező az ellenőrző összeg használata, mert ott nincs a fejrészben ellenőrző összeg.

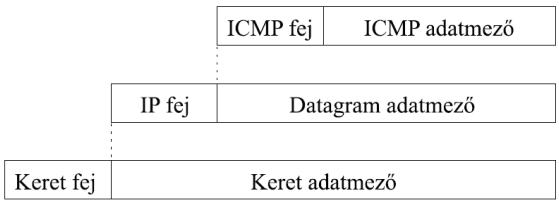
**data** adatok

Itt található beágyazva az UDP fölötti protokoll adategysége.

Amint említettük, az UDP nem nyújt megbízható átvitelt. Amit nyújt, az egyrészt a végpontok azonosítása a portok segítségével és erre építve multiplexálás/demultiplexálás, másrészt a fent említett hibavédelem, amely az UDP adategységen túl az IP-címekre is kiterjed.

# Internet Control Message Protocol

Az *Internet Control Message Protocol*t az RFC 792-ben definiálták. Bár az ICMP az IP rétegre építve szállítja az IP szolgálati közleményeit, mégsem tekinthető egy magasabb szintű (a TCP-vel és az UDP-vel egy szinten levő, azaz szállítási) protokollnak. Éppen a funkciója miatt kötelező része minden IP implementációnak. Üzenetei viszont a szállítási protokollok üzeneteivel azonos módon ágyazódnak be: 13. ábra.

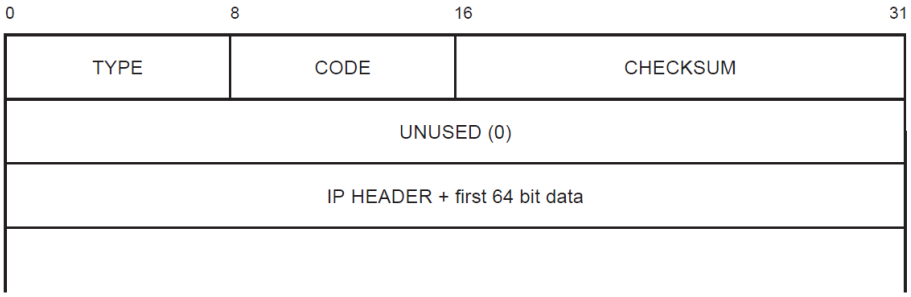



*13. ábra. Az ICMP adategységének beágyazása*

A hálózat felhasználói számára az ICMP üzenetek általában észrevétlenek, a felhasználó csupán egy jól működő hálózatot lát. Van azonban néhány hálózat-karbantartással kapcsolatos feladat, amikor a felhasználók is igénybe vehetik az ICMP szolgáltatásait, ezekkel hamarosan fogunk találkozni, de előbb még megismerjük az ICMP üzenetek formátumát és néhány fontosabb ICMP üzenetet.

## ICMP üzenetformátum

Az ICMP üzenetek formátuma egyedi. Ami közös bennük, az az ICMP fejrész első 32 bitjén található 3 adatmező. A további rész kiosztása függ az üzenet típusától. Példaként nézzük meg egy tipikus hibaüzenetnek számító ICMP üzenet felépítését a 14. ábrán.




*14. ábra. A Destination Unreachable ICMP üzenet felépítése*

Az általános, minden ICMP üzenetre jellemző mezők:

**Type** típus

Ez a 8 bites szám adja meg, hogy melyik ICMP üzenetről van szó.

**Code** kód

Ennek a 8 bites számnak az értelmezése az ICMP üzenet típusától függ. Gyakran az adott típusú üzenet valamely altípusát jelenti, de az is lehet, hogy érdemben nem használjuk, ekkor az értéke 0.

**Checksum** ellenőrző összeg

Ugyanolyan algoritmussal képzett 16 bites ellenőrző összeg, mint az IP-nél.

A további mezők tehát üzenettípusonként eltérőek. Több esetben is előfordul kihasználatlan mező, ezt a forrásnak csupa 0 értékű bitekkel kell feltöltenie, a címzettnek pedig figyelmen kívül kell

hagynia az értékét. Amennyiben hibaüzenetről van szó, az ICMP üzenet tartalmazza még a hibaüzenetet kiváltó IP datagram fejrészét és az adatrész első 64 bitjét. Vegyük észre, hogy az IP réteg *nem tudja*[13], hogy fölötte mi utazik, de akár TCP, akár UDP, ez a 64 bit tartalmazza a hiba kezeléséhez szükséges információt (pl. portszámokat).

Megjegyzés: ICMP hibaüzenettel kapcsolatos hiba következtében nem keletkezhet újabb hibaüzenet. Ez fontos védelem a hibaüzenetek öngerjesztő elszaporodása ellen, viszont ez azt is maga után vonja, hogy ICMP hibaüzenetek nyom nélkül elveszhetnek. Ezt természetesen tolerálni tudjuk, hiszen az IP réteg amúgy sem megbízható.

## Fontosabb ICMP üzenetek

Ismerjük meg a fontosabb ICMP üzeneteket! Az egyes üzeneteknél zárójelben megadjuk azok típusszámát is. A felsorolásban az RFC 792-beli sorrendet követjük.

**Destination Unreachable** (3) cél nem elérhető

Ennek több oka lehet, amiket a Code mező tartalma különböztet meg:

**0 = net unreachable** a célhálózat nem elérhető

**1 = host unreachable** a célgép nem elérhető

**2 = protocol unreachable** a kívánt (IP fölötti) protokoll nem elérhető

**3 = port unreachable** a kért port nem elérhető

**4 = fragmentation needed and DF set** a datagram tördelésére lenne szükség, de a DF (do not fragment) bit be van állítva

**5 = source route failed** a *forrás általi útválasztás* sikertelen

**Time Exceeded** (11) időtúllépés

Ha egy datagram TTL-je lejár (0-ra csökken), akkor az az útválasztó, ahol éppen tartózkodik, köteles (MUST) eldobni a datagramot. Ilyenkor ezzel az üzenettel jelezheti (MAY) a forrás számára, hogy eldobta. Tördelt datagram esetén a célállomás összerakáskor eldobja a töredéke(ke)t, ha a TTL lejár, ilyenkor ugyancsak ezzel az üzenettel jelezheti a forrás számára a hibát.

**Parameter Problem** (12) érvénytelen fejrészmező

Ha egy útválasztó vagy egy számítógép nem tudja értelmezni egy datagram fejrészének valamely mezőjét (például Type of Service vagy valamilyen opció), és ennek következtében eldobja a datagramot, ilyen üzenettel jelezheti azt a forrás számára. Ez az ICMP üzenet az első 32 bitje után tartalmaz egy 8 bites pointert. Ha a Code mező értéke 0, akkor ez a pointer az eredeti datagramnak arra az oktettjére mutat, amely a problémát okozta. Természetesen az eredeti datagram fejrésze és az első 64 adatbitje is része az üzenetnek (32 bites határra igazítottan elhelyezve).

**Redirect** (5) átirányítás

Egy router megadhat egy állomásnak egy rövidebb utat a datagramban szereplő *cél* felé, de ettől még továbbítja az eredeti datagramot. (A háttérben az van, hogy a routerekről feltételezzük, hogy pontosan tudják, merre vezet a rövidebb útvonal, míg az állomások nem, de így megtanulhatják.) A kód mező fejezi ki, hogy pontosan mit is értünk cél alatt:

[13] A Protocol mezőben persze benne van az IP adatrészében szállított protokoll azonosítója, de az IP-nek nem dolga, hogy az adatmezőjének tartalmát értelmezze.

**0 = Network** hálózat

**1 = Host** állomás

**2 = Type of Service and Network** szolgáltatástípus és hálózat

**3 = Type of Service and Host** szolgáltatástípus és állomás

**Echo** (8) visszhang kérés

A forrás arra kéri a címzettet, hogy küldje vissza az üzenetet. A minden ICMP üzenetben azonos funkciójú első 32 bit után ez az üzenet tartalmaz egy 16 bites azonosítót, és egy 16 bites sorszámot. Ez utóbbi az egymást követő üzenetekben egyesével nő.

Megjegyzés: bár az RFC 792 csak „Echo”-nak nevezi, a Wiresharkban „Echo request” néven találkozunk vele.

**Echo Reply** (0) visszhang válasz

Az Echo üzenet címzettje ezzel az üzenettel válaszol. A válasz mezői általában megegyeznek a kérés mezőivel, de természetesen a típus mező nem, és az ellenőrző összeget is újra ki kell számítani.

**Timestamp** (13) időbélyeg kérés

Az Echo üzenetet további három, egyenként 32 bites mezővel bővíti:

**Originate Timestamp** küldési időbélyeg

**Receive Timestamp** vételi időbélyeg

**Transmit Timestamp** visszaküldési időbélyeg

Az időbélyegek az UTC szerint éjfél óta eltelt időt tartalmazzák ms-ban mérve.

**Timestamp Reply** (14) időbélyeg válasz

Válasz a Timestamp üzenetre. A válaszoló a mezőket az *Echo Reply*hoz hasonlóan másolja, illetve értelem szerint kitölti.

## Felhasználók számára is elérhető ICMP üzenetek

Van két olyan hálózati tesztelésre szolgáló parancs (segédprogram), ami a legtöbb operációs rendszeren megtalálható (esetleg más néven).

### A ping parancs

A *ping* parancs egy vagy több *visszhang kérés* ICMP üzenetet küld a felhasználó által megjelölt másik gépre. A címzett pedig *visszhang válasz* ICMP üzenetet küld annak a gépnek, ahonnan a kérés érkezett. A felhasználó ilyen módon tesztelheti, hogy egy másik gép elérhető-e a hálózaton keresztül.

A ping parancs célszerűen kiírja a visszaérkező üzenetből a TTL értékét, így azt is megtudhatjuk, hogy milyen távol van a vizsgált gép (útválasztók számában mérve).

A ping parancs ezenkívül mérni szokta a *visszhang kérés* küldése és a vele azonos sorszámú *visszhang válasz* megérkezése között eltelt időt, így megtudhatjuk a teljes oda-vissza út idejét (RTT, *round-trip time*).

## A traceroute parancs

A *traceroute* parancs egy távoli géphez vezető útvonal során érintett útválasztók válaszidejét[14] deríti ki. Ehhez többféle trükkös módszert is használhat, például egy elterjedt megoldás, hogy a vizsgált eszköz (router) *33434-es UDP portjára*[15] küld egy UDP csomagot. A beállított TTL-t 1-ről növeli. Amíg a TTL túl kicsi, addig *time exceeded* üzenetet kap vissza valamelyik közbenső routertől, amikor pedig már elég nagy, akkor *port unreachable* üzenet érkezik (feltéve, hogy a 33434-es porton nem figyel alkalmazás, ha mégis, akkor a felhasználó választhat más portot). Alternatívaként a felhasználó azt is megadhatja, hogy UDP datagram helyett ICMP *echo* üzenetet küldjön.

Megjegyzés: Windows 7 alatt a *tracert* parancs ICMP *echo* üzenetet küld.

[14] Az időben benne van a teljes oda-vissza út ideje (RTT – round-trip time), valamint a válasz előállításának az ideje, ami erősen függ a router terheltségétől.
[15] Egy várhatóan nem használt célport, szükség esetén lehet változtatni, BSD-ben lehet TCP-t is használni.

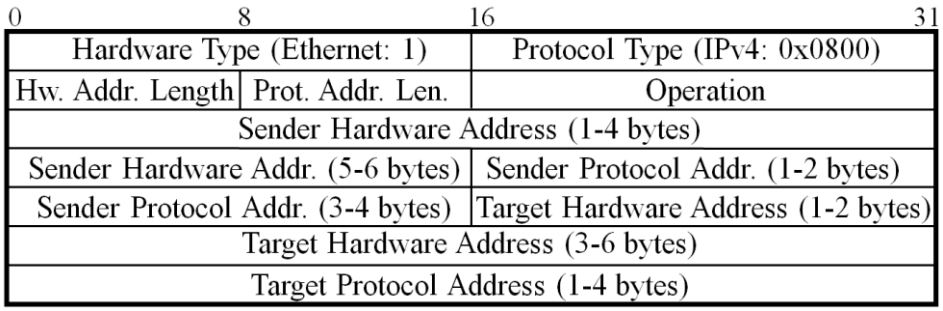
# Address Resolution Protocol

Az IPv4 négy oktett méretű címeket használ az állomások azonosítására, míg a fizikai-adatkapcsolati szintű hálózati megvalósításoknak is megvan a saját címzési rendszere. Például az Ethernet 6 bájtos címeket használ. Az IPv4[16] négy oktettjéről a 6 bájtos MAC címre való leképzést hívjuk *címfeloldás*nak, amit az *Address Resolution Protocol* (ARP, RFC 826) segítségével valósítunk meg. Ennek a működéséhez fontos körülmény, hogy mindig csak olyan állomások IP-címéhez kell kiderítenünk a hozzájuk tartozó MAC címet, ami velünk azonos hálózaton (broadcast domainben) van.

Az ARP üzenetek az Ethernet keret adat mezőjében utaznak. Ethernet szinten a célcím *ARP Request* esetén broadcast (FF:FF:FF:FF:FF:FF), *ARP Reply* esetén általában a kérés küldőjének unicast címe, de elvileg lehet broadcast is. Az *EtherType* mező értéke mindig 0x0806, erről ismerhető fel, hogy az Ethernet fölött ARP üzenet utazik.

## Az ARP üzenetek felépítése

Az ARP üzenet felépítését a 15. ábrán mutatjuk be.




*15. ábra. Az ARP üzenetek felépítése*

Ahol az egyes mezők értelmezése és értéke (az utolsó négy esetén az értékek a működésnél):

**Hardware Type** hardver típusa

A fizikai-adatkapcsolati hálózati megvalósítás típusa. Ethernet esetén az értéke 1.

**Protocol Type** hálózati protokoll típusa

A hálózati protokollt azonosító számérték. Az EtherType esetén használatos azonosítók használhatók itt is, kivéve az IPv6-ot, mert az nem ARP-t használ. IPv4-nél az értéke: 0x0800.

**Hardware Address Length** hardver címhossz

A fizikai-adatkapcsolati hálózati megvalósítás címhossza. Ethernet esetén az értéke 6.

**Protocol Address Length** hálózati protokoll címhossz

A hálózati protokoll címhossza. IPv4 esetén az értéke 4.

**Operation** művelet

*ARP Request* esetén 1, *ARP Reply* esetén 2.

**Sender Hardware Address** (SHA) küldő hardver címe

**Sender Protocol Address** (SPA) küldő hálózati címe

**Target Hardware Address** (THA) kérdéses/cél/megcélzott hardver cím

[16] Az IPv6-ban teljesen más megoldást használnak egy IP-címhez tartozó MAC-cím kiderítésére.

**Target Protocol Address** (TPA) kérédéses/cél/megcélzott hálózati cím

## Címfeloldás ARP segítségével

Az „A” állomás szeretné megtudni a „B” állomás MAC-címét annak IP-címe alapján.

1. „A” Ethernet szinten az FF:FF:FF:FF:FF:FF (broadcast) célcímre küld egy *ARP Request*et, melyben a forráscím a sajátja, az EtherType mező értéke 0x0806. Az *ARP Request* (nem triviális) mezői:
   - Operation: 1 (Request)
   - SHA: <„A” MAC-címe> (Megegyezik a keret fejrészében találhatóval.)
   - SPA: <„A” IP-címe>
   - THA: 00:00:00:00:00:00 (ismeretlen) (De a keret fejrészében a cél MAC-cím: FF:FF:FF:FF:FF:FF !!!)
   - TPA: <„B” IP-címe>
2. Az *ARP Request* üzenetet a *broadcast domain* összes állomása veszi, és tárolja az „A” *IP-cím – MAC-cím* párosát az *ARP Cache* táblájában.
3. „B” felismeri a saját IP-címét az *ARP Request* üzenetben, ezért válaszként „B” Ethernet szinten az „A”-nak címezve[17] küld egy *ARP Reply*t, melyben a forráscím a sajátja, az EtherType mező értéke most is 0x0806. Az *ARP Reply* (nem triviális) mezői:
   - Operation: 2 (Reply)
   - SHA: <„B” MAC-címe> (Megegyezik a keret fejrészében találhatóval)
   - SPA: <„B” IP-címe>
   - THA: <„A” MAC-címe> (Megegyezik a keret fejrészében találhatóval)
   - TPA: <„A” IP-címe>
2. „A” veszi a választ, és eltárolja „B” *IP-cím – MAC-cím* párosát.

## Az ARP Cache tábla

Az állomások bizonyos ideig tárolják az ARP-vel megszerzett címfeloldási információkat. Mivel az *ARP Request*et adatkapcsolati szinten broadcast címre kell küldeni, a küldő *IP-cím – MAC-cím* párosát minden állomás el tudja tárolni. Ha esetleg a választ is broadcast címre küldik, akkor azt is el lehet tárolni. A dinamikus bejegyzéseken kívül (az **arp** menedzsment szoftverrel) statikus bejegyzések is felvehetők. Az *ARP Cache* tartalma általában az **arp -a** paranccsal meg is jeleníthető, az **arp -d** paranccsal pedig bejegyzések törölhetők belőle.

Az ARP Cache táblában kétféle típusú bejegyzések lehetnek:

- **Statikus**: Manuálisan felvitt bejegyzés eredménye. Amíg nem törlik, változatlanul marad.
- **Dinamikus**: ARP címfeloldás eredménye. Gyorsítótár (cache) funkciót valósít meg; ne kelljen mindig lekérdezni. Egy idő után elévül és törlődik.

Az ARP Cache tábla elvi felépítése:

[17] De az is lehetséges, hogy a „B” a választ broadcast címre küldi.

| IP-cím | HW-cím | típus |
|---|---|---|
| <IP-cím1> | <MAC-cím1> | statikus |
| <IP-cím2> | <MAC-cím2> | dinamikus |

A gyakorlatban még egyéb információt is tárolnak, például a hardver típusát (ami Etherneten kívül más is lehet) és az interfészt, amelyiken keresztül az a hálózat elérhető, amelyen az adott szomszéd található. Például Linux alatt az ARP Cache táblát megvizsgálva:

```
root@dev:~# arp -n
Address                  HWtype  HWaddress           Flags Mask            Iface
193.224.130.161          ether   00:15:17:54:99:78   C                     eth0
```

(A **C** flag jelzi a cache-elt (dinamikus), az **M** pedig a manuálisan beállított (statikus) értékeket.)

## IPv4 Address Conflict Detection

Mielőtt egy host elkezd egy IP-címet használni, meg kell (SHOULD) vizsgálnia, hogy az IP-cím nincs-e már használatban. Erre való az *ARP Probe* üzenet: ez egy speciális *ARP Request*, amellyel a használni kívánt IP-címhez tartozó MAC-címre kérdez rá a TPA mezőben, de az SPA mezőben a 0.0.0.0 IP-cím található; ennek célja, hogy ne szennyezze mások ARP Cache-ét, azaz ha mégsem használhatja a címet, akkor ne tárolják el a hamis információt. Ha az *ARP Probe* üzenetre választ kap, akkor tudja, hogy más valaki már használja a kérdéses IP-címet.

Ha DHCP-vel[18] kapott IP-címről derül ki, hogy más valaki már használja, akkor kötelező (MUST) a DHCP szerver felé DHCPDECLINE üzenetet küldeni.

Az RFC 5227 nem rendelkezik róla konkrétan, hogy az *ARP Probe* üzenetet hányszor kell elküldeni, de megemlíti, hogy a megfelelően alacsony hibavalószínűség érdekében többször.

Ha az IP-cím szabadnak bizonyult, akkor a fentiek szerint eljáró host köteles (MUST) *ARP Announcement* üzenettel jelezni mindenki számára, hogy az adott IP-címet ő fogja használni. Ez olyan *ARP Request* típusú üzenet, ahol az SPA és a TPA mezőben egyaránt az adott IP-cím szerepel. Mivel broadcast címre küldik, mindenki megkapja, és az ARP Cache tábláját frissíteni tudja.

Sajnálatos módon *ARP Probe* helyett bizonyos implementációkban *Gratuitous ARP* (kéretlen ARP) üzeneteket használnak. Így hívják mind az *ARP Request* nélkül, broadcast címre küldött *ARP Reply* üzeneteket, mind az *ARP Probe* nélkül küldött *ARP Announcement* üzeneteket.

Ez a módszer azért nem jó, mert:

- Nem óvja meg a már működő gépek működőképességét.
- Nem teszi lehetővé a most induló gépnél sem azt, hogy automatikusan (emberi beavatkozás nélkül) más IP-címet használjon.

[18] A DHCP leírása a következő fejezetben található.

# Dynamic Host Configuration Protocol

## A DHCP általános jellemzői

A DHCP az alkalmazási rétegben működő protokoll. Segítségével a hostok automatikusan juthatnak hozzá a kommunikációjukhoz szükséges hálózati azonosítókhoz: IP-cím, hálózati maszk, alapértelmezett átjáró, stb. A DHCP-t az BOOTP kiterjesztéseként definiálták. A DHCP aktuális definíciója megtalálható: RFC 2131.

## A DHCP lehetőségei

- IP-címek kiosztása MAC-cím alapján DHCP szerverrel
- Szükség esetén (a DHCP szerveren előre beállított módon) egyes kliensek számára azok MAC-céméhez fix IP-cím rendelhető.
- IP-címek kiosztása dinamikusan
- A DHCP szerveren beállított tartományból „érkezési sorrendben” kapják a kliensek az IP-címeket. Így elegendő annyi IP-cím, ahány gép egyidejűleg működik.
- Az IP-címeken kívül további szükséges hálózati paraméterek is kioszthatók:
    - Hálózati maszk
    - Alapértelmezett átjáró
    - Névkiszolgáló
    - Domain név
    - Hálózati rendszerbetöltéshez szerver és fájlnév

Az IP-címek bérlésének szabályai:

- A DHCP szerver a klienseknek az IP-címeket bizonyos *bérleti időtartam*ra (lease time) adja oda használatra.
- Az időtartam hosszánál a szerver figyelembe veszi a kliens esetleges ilyen irányú kérését.
- Az időtartam hosszát a szerver beállításai korlátozzák.
- A bérleti időtartam lejárta előtt a bérlet meghosszabbítható.
- Az IP-cím explicit módon vissza is adható.

## A DHCP működése és üzenetei

A kliens és a szerver *DHCP üzenetek*kel kommunikálnak. A DHCP üzenetek BOOTP üzenetekben opcióként jelennek meg. A BOOTP üzenetek IP fölött, UDP-be ágyazva haladnak. Amíg a kliensnek nincs érvényes IP-címe, addig 0.0.0.0-t használ. Broadcast esetén IP szinten természetesen 255.255.255.255 címre küldi az üzenetet (Ethernet szinten pedig FF:FF:FF:FF:FF:FF címre). UDP-ben a kliens portszáma: 68, a szerveré: 67.

A továbbiakban a DHCP üzenetek neve mellett feltüntetjük, hogy ki küldi kinek: „küldő → címzett” formában. Jelölések:

- K: kliens
- S: szerver

- B: broadcast (IP és Ethernet szinten is)

Egy IP-cím megszerzéséhez a következő négy üzenetre van szükség:

**DHCPDISCOVER K→B**

Egy kliens küldi broadcast címre, hogy feltérképezze az elérhető DHCP szervereket és ajánlataikat. A kliens opcionálisan (nem az IP fejrészben, hanem DHCP opcióként) megadhatja a legutoljára használt IP-címét, de ez NEM azonos a bérlet meghosszabbításával!

**DHCPOFFER S→K**

Egy DHCPDISCOVER üzenetre egy vagy több szerver válaszol, megadja, hogy milyen IP-címet és paramétereket tud kínálni. Ekkor ezek a kliens számára még NEM használhatók!

**DHCPREQUEST K→B**

A kliens ezzel az üzenettel egyidejűleg elfogadja valamely szerver ajánlatát, és implicit módon elutasítja a többiekét (broadcast miatt minden szerverhez eljut). A kliens megjelölheti benne a kért bérleti időtartamot is.

**DHCPACK S→K**

A szerver ekkor megerősíti a kliensnek az IP-cím bérletét, és megadja, hogy milyen időtartamra kapja meg a kliens.

A kliens utána a bérleti idő lejártáig használhatja az IP-címet, de a címütközés elkerülése érdekében erősen ajánlott (SHOULD) *ARP Probe* segítségével ellenőriznie, hogy más nem használja-e.

- Ha más nem használja, a kliens *ARP Announcement*tel kihirdeti, hogy az övé.
- Ha más használja, a kliens DHCPDECLINE-nal jelzi a DHCP szervernek, és természetesen másik IP-címet kér.

Kedvezőtlen esetben az alábbi két üzenet fordulhat elő:

**DHCPNAK S→K**

Ezzel az üzenettel jelzi a szerver, hogy a kliens kérése nem teljesíthető.

**DHCPDECLINE K→S**

A kliens jelzi a szervernek, hogy az adott IP-címet már más használja.

A bérleti idő lejártán belül a kliens hosszabbítást kérhet, ekkor nem kell az egész folyamatot lejátszania, elegendő:

**DHCPREQUEST K→S**

Az üzenet küldésekor a kliens még használja az érvényesen bérelt IP-címét és a kérést nem broadcast címre, hanem a szervernek küldi.

**DHCPACK S→K**

A szerver ekkor meghosszabbítja kliensnek az IP-cím bérletét és megadja, hogy milyen időtartamra kapja meg a kliens.

Természetesen ilyenkor a kliensnek nem kell további ellenőrzést végeznie, hiszen ezt a címet már eddig is használta. Így most a DHCPDECLINE üzenet sem jön szóba, de a szervertől most is kaphat DHCPNAK üzenetet.

A bérleti idő lejártán belül a kliens korábban is visszaadhatja (MAY) az IP-címet:

**DHCPRELEASE K→S**

Ezzel a kliens lemond a hátralevő bérleti időről, a szerver újra kioszthatja a címet.

Amennyiben egy kliensnek már van IP-címe (például statikusan be van állítva), akkor is kérhet más paramétereket:

**DHCPINFORM K→S**

Ezt a kliens a szervernek unicast üzenetként küldi.

Válaszul a szerver ugyanígy unicast üzenetként küldi egy DHCPACK üzenetben a további hálózati beállításokat. Ilyenkor a szerver nem ellenőrzi, hogy a kliens rendelkezik-e érvényes IP-cím bérlettel.

## Wireshark

A Wireshark protokollanalizátor ismertetése önmagában is meghaladná e segédlet kereteit. Azonban a felhasználói felülete nagyon intuitív, így a hallgatók önállóan megismerkedhetnek az alapokkal az 1. mérés 1. feladatának megoldása közben, és az 1. mérés végére már képesek lesznek a segítségével hálózati problémák önálló megoldására (ezt teszteli a 10. feladat).

Felkészítésként csak a szűrők használatát említjük meg. Szűrésre kétféle esetben is lehetőség van:

- A hálózati forgalom lehallgatásakor, erre való a *csomagelkapási szűrő* (capture filter).
- A már lehallgatott forgalom megjelenítésénél, erre való a *megjelenítési szűrő* (display filter).

Az elsőt csak röviden érintjük az 1. mérés közben, a másodikat viszont gyakran fogjuk használni. Ezért az utóbbiról adunk egy nagyon rövid bevezetőt.

A megjelenítési szűrő szintaktikája és szemantikája is nagyon intuitív, és alapvető C programozási ismeretek birtokában igen könnyen megtanulható. Ráadásul a Wireshark gépelés közben megadja a kulcsszavak lehetséges folytatását, sőt még „syntax higlight" funkciót is nyújt számunkra: ha szintaktikailag helyes az éppen begépelt szűrő, akkor a háttere zöld; ha helytelen, akkor rózsaszín, ha pedig félreérthető (lásd később), akkor sárga.

A kifejezések építésének szabályai:

- Ha megadunk egy protokollnevet, amely a protokollhierarchia bármely szintjén lehet (pl. **`eth`**, **`ip`**, **`arp`**, **`tcp`**, **`bootp`**, **`dns`**, **`http`**, **`ssh`**), akkor a csomaglistában azok a csomagok jelennek meg, amelyekben szerepel az adott protokoll.
- Protokollok mezőit a protokollnév.mezőnév alakban adhatjuk meg. Például: **`eth.type`**, **`ip.ttl`**, **`tcp.seq`**, stb.
- Egy mező értéke a C nyelvben használatos összehasonlító operátorokkal vizsgálható: **`==`**, **`!=`**, **`<`**, **`>`**, **`<=`**, **`>=`**. Például: **`ip.ttl>10`**. És ugyanúgy használhatók a shell scriptekben használatos angol nyelvű kétbetűs rövidítések is: **`eq`**, **`ne`**, **`gt`**, **`lt`**, **`ge`**, **`le`**. Például: **`tcp.ack eq 1000`**.
- Egy és kétoperandusú logikai operátorok is használhatók, így C stílusban: **`!`**, **`&&`**, **`||`**, **`^^`** (az utóbbi jelentése: XOR). Például: **`ip.src==10.0.1.1 || ip.src==192.168.1.1`**, Shell script stílusban: **`not`**, **`and`**, **`or`**, **`xor`**. Például: **`not arp`**.
- Nagyon hasznos a substring operator: szögletes zárójelen belül adható meg egy n:m tartomány. Például **`eth.src[:3]==00:c0:aa`**. Ennek lehetőségei lényegesen bővebbek, érdemes megnézni a dokumentációt [9].
- További kényelmi lehetőség, hogy a forrás és célcímek bármelyikét jelenti az **`addr`** mezőnév. Tehát például az **`(ip.src==10.1.1.1)||(ip.dst==10.1.1.1)`** egyszerűbben írható úgy, hogy: **`ip.addr==10.1.1.1`**. Hasonlóan használható a **`port`** mezőnév is TCP-nél és UDP-nél. Például a **`tcp.port==80`** minden olyan TCP szegmensre illeszkedik, ahol a forrás vagy a cél port szám 80.
- Óvatosnak kell viszont lenni a **`!=`** operátorral. A Wireshark dokumentációja [9] a következő példát hozza: **`ip.addr!=1.2.3.4`**. Ez NEM azt jelenti, amit első pillanatra gondolnák, hogy „sem a forrás sem a cél IP-címe: 1.2.3.4", hanem azt, hogy „van olyan IP-címe, ami nem 1.2.3.4". Az első leírásnak megfelelő szűrő így néz ki: **`!(ip.addr==1.2.3.4)`**. (A Wireshark sárga háttérrel jelzi, ha egy szűrőt félreérthetőnek talál. Érdemes ilyenkor újra átgondolni, mit is jelent, amit leírtunk!)

A Wiresharkkal eredményes ismerkedést és a mérés során hatékony tanulást kívánunk!

## Ajánlott irodalom

Az ajánlott irodalom célja az, hogy az egyes témakörök iránt mélyebben érdeklődő hallgatóknak legyen hol tájékozódniuk. Elolvasásuk a mérésre való felkészüléshez nem szükséges.


[1] Lencse Gábor: *Számítógép-hálózatok*, Universitas-Győr Nonprofit Kft, Győr, 2008.
[2] Alexis Ferrero: *Az örök Ethernet*, Szak Kiadó Kft., Bicske, 2001.
[3] IEEE Standards Association, *MA-L Public Listing*, http://standards.ieee.org/develop/regauth/oui/public.html
[4] Douglas E. Comer, *Internetworking with TCP/IP*, Vol. I: Priciples, Protocols and Architectures, 3rd ed., Prentice Hall, Inc., Upper Saddle River, NJ, 1995.
[5] W. Richard Stevens, *TCP/IP Illustrated*, Vol. 1. The Protocols, Addison Wesley Longman, Reading, MA, 1994.
[6] Dave Roberts, *Internet Protocols Handbook*, The Coriolis Group, Inc., Scottsdale, AZ, 1996.
[7] Stephen A. Thomas: *IP kapcsolás és útvonalválasztás*, Kiskapu Kft, Budapest, 2002.
[8] Raj Jain, *FDDI Handbook: High Speed Networking Using Fiber and Other Media*, Addison–Wesley Publishing Company, Reading MA, April 1994.
[9] Ulf Lamping, Richard Sharpe, Ed Warnicke, *Wireshark Users's Guide*, https://www.wireshark.org/docs/wsug_html_chunked