INFOKOMMUNIKÁCIÓS SZOLGÁLTATÁSOK ÉS ALKALMAZÁSOK

Transzport protokollok I. / II.

Szabó Sándor
Huszák Árpád
BME Híradástechnikai Tanszék
szabos@hit.bme.hu

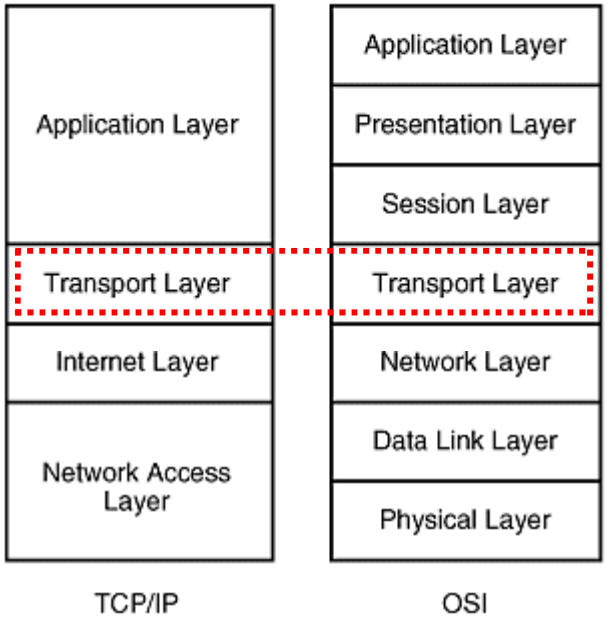
2011. május 11.,
Budapest

Kivonat

- ISO/OSI és TCP/IP architektúra
  - Szállítási réteg
- Alkalmazások típusai
- Vezeték nélküli hálózatok jellemzői
- Protokollok
- Mobilitás támogatás a szállítási rétegben
- TCP
  - protokoll ismertetése
  - torlódásszabályozás
  - TCP vezeték nélküli környezetben
  - TCP variánsok mobilitás támogatására:
    - Indirect-TCP
    - Snoop TCP
    - Mobile TCP (M-TCP)
    - Fast retransmit/fast recovery
    - Transmission/time-out freezing
- UDP

ISO/OSI és TCP/IP modell

- International Standards Organization (Nemzetközi Szabványügyi Szervezet)
  - Open System Interconnection (nem szabvány, hanem csak egy ajánlás)
- A számítógép hálózatok a megvalósításuk bonyolultsága miatt rétegekre osztódnak
  - Mik legyenek az egyes rétegek feladatai és azok határai hol legyenek?



Szállítási réteg (transport layer)

- A szállítási réteg különböző típusú szolgáltatásokat nyújthat a felsőbb rétegek felé
  - hogy milyen típusú szolgáltatásra van szükség, az alkalmazás típusától függ
- A szállítási réteg jellemzői:
  - A transzport réteg takarja el az alatta levő hálózati architektúrától függő részleteket az alkalmazások elől
  - A hálózati topológiát nem ismeri, csak a két végpontban van rá szükség.
- Szolgáltatások:
  - A szállítási szolgáltatás átlátszó, megbízható és költséghatékony adatátvitelt végez a viszonyentitások között.
  - Megbízható kommunikációs csatornát biztosíthat a felette levő protokollrétegeknek
  - Feladata a végpontok közötti hibamentes adatátvitel biztosítása, amennyiben szükség van rá

Szállítási réteg (transport layer)

- A viszonyrétegnek nyújtott szolgáltatások:
  - a viszonyentitások egyértelmű azonosítása szállítási címükkel (port cím)
  - szállítási összeköttetések létesítése, fenntartása és bontása
  - átlátszó adatátvitel (normál és gyorsított)
  - a megválasztott szolgáltatásminőség fenntartása
  - összeköttetések nyalábolása és hasítása

Szállítási réteg (transport layer)

- Szállítási összeköttetések felépítése, bontása, csomagok sorrendbe állítása
- Szolgálati primitívek az OSI-környezetben, és a TCP/IP környezetben
  - Egy szolgálatot bizonyos alapmûveletek (primitívek) segítségével írhatunk le. Ezekkel definiáljuk, hogy egy szolgálat milyen tevékenységet végez el, és milyen jelzést ad tovább egy másik primitívnek.

OSI:	TCP/IP:
•request •indication •response •confirm	•listen •connect •send •receive •close

Szállítási réteg (transport layer)

- Adatátviteli szolgáltatástípusok:
  - *Normál mód*: a küldő hoszttól érkező adat egy várakozó sorba kerül, innen később továbbításra kerül.

    FIFO – First-In-First-Out
  - *Sürgősségi mód*: a sor következő tovább küldésre kerülő adata helyére kerül és a lehető leghamarabb továbbküldik.

    LIFO – Last-In-First-Out

Szállítási réteg

- A hálózati réteg csak 1db címezhető kommunikációs végpontot biztosít hálózati csatlakozónként, addig a transzport réteg feladata az ennél több címezhető egység biztosítása is.
- Erre azért van szükség, mert egy számítógépen több program is futhat
  - egyidejűleg több is akarhat a hálózaton keresztül más alkalmazásokkal kommunikálni (ekkor fontos, hogy a kommunikáló partnerek csomagjai ne keveredjenek egymással)
- Az Internet hálózatban használt transzport protokollok:
  - TCP
  - UDP/UDP-Lite
  - DCCP/DCCP-Lite
  - SCTP
  - (RTP, RTCP, RTSP)

Alkalmazások típusai

- Késleltetésre nem érzékeny, bithibára igen
  - Interaktív (Telnet)
  - Adat letöltés (HTTP, FTP)
  - Ajánlott transzport protokoll: TCP, SCTP
- Késleltetésre érzékeny, bithibára kevésbé
  - Streaming (video, audio)
  - Hangátvitel
  - Ajánlott transzport protokoll: UDP, DCCP

Az alkalmazás típusától függően kell kiválasztani a megfelelő szállítási rétegbeli protokollt.

Alkalmazások típusai

Alkalmazás	Alkalm. réteg protokollja	Szállítási réteg
e-mail	SMTP	TCP
távoli hozzáférés	Telnet	TCP
Web	HTTP	TCP
file átvitel	FTP	TCP
távoli file server	NFS	UDP
Multimédia streaming	egyedi	UDP, DCCP, SCTP
IP telefónia	egyedi	UDP, DCCP
hálózat menedzsment	SNMP	UDP
útvonalválasztás - routing	RIP	UDP

Vezeték nélküli, mobil hálózatok

- A vezeték nélküli közeg használta esetén számos olyan problémával kell megbirkózni, amelyek vezetékes hálózatoknál nem jelentkeznek:
  - korlátozott sávszélesség
  - sokkal megbízhatatlanabb átvitel, csatornahiba
  - nagy zavarérzékenység
  - lehallgathatóság
  - dinamikus topológia
  - jelentős késleltetés és késleltetés-ingadozás (jitter)
  - handover – adminisztratív üzenetek - késleltetés

Mobilitás támogatás

- Sok probléma merült fel annak köszönhetően, hogy az egyes rétegek elég lazán vannak definiálva
- Egyes szolgáltatások több rétegben is megvalósításra kerültek, mások pedig egyikben sem
- A mobilitás egyik réteghez sem tartozik egyértelműen
- A mobilitást megvalósító rendszerek követelményei:
  - Átlátszó átvitel
  - Lokáció menedzsment
  - Infrastruktúra mentesség

Mobilitás támogatás

- Átlátszó átvitel

  Hálózatok közötti váltás ne okozzon nagy adatvesztést, a váltás ne tartson sokáig és a hosszú távú kapcsolat orientált protokollokat használó programok zavartalanul futhassanak tovább

- Lokáció menedzsment

  A mobil eszköznek mindvégig elérhetőnek kell lennie egy statikus azonosító segítségével függetlenül attól, hogy helyileg éppen hol van.

- Infrastruktúra mentesség

  Minél jobban a hálózat szélén van a mobilitás megvalósítva, annál kevesebb változtatásra van szükség a jelenlegi hálózatokban.

Protokollok

- Megbízható:
  - TCP és variánsai
  - SCTP
- Nem megbízható:
  - UDP és UDP-Lite
  - DCCP

Finomhangolás a *sysctl* használatával

- A *sysctl* egy olyan felület, amely lehetőséget biztosít egy működő Linux/FreeBSD rendszer megváltoztatására.
- Segítségével hozzáférhetünk az operációs rendszer különböző paramétereihez
  - A kernel rengeteg apró opciójához hozzáférhetünk, melyek megfelelő beállításával egy tapasztalt rendszergazda kezében drasztikusan növelhető a rendszer teljesítménye.
  - A *sysctl* alkalmazásával több mint 500 rendszerszintű változó kérdezhető le és állítható be.
  - A /proc/sys/ könyvtárban található beállításokat olvassa és írja
- sysctl funkciói:
  - a rendszer beállításainak lekérdezése
  - módosítása

Finomhangolás a *sysctl* használatával

- Az összes lekérdezhető változó megjelenítése:

```
sysctl -a
```

- Szűrés pl. TCP változókra:

```
sysctl -a | grep tcp
```

- Változó módosítása:

```
sysctl -w kernel.domainname="example.com„
sysctl -w net.ipv4.tcp_congestion_control=bic
```

- Config file betöltése:

```
sysctl -p /etc/sysctl.conf
```

TCP

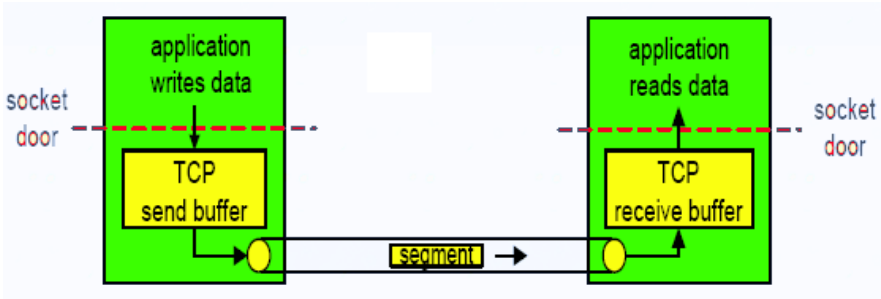
- Transmission Control Protocol [RFC-793]
- 1981
- Az egyik leggyakrabban használt transzport protokoll
- szabványt vezetékes hálózatra dolgozták ki, azonban a ma egyre szélesebb körben használt vezeték nélküli hálózatok karakterisztikái jelentősen különböznek vezetékes hálózatok adatátviteli tulajdonságaitól.
- olyan vezetékes összeköttetésekre dolgozták melyeknek a jellemzőik a következők:
  - nagy sávszélesség
  - kis késleltetés
  - kis hibavalószínűség

TCP jellemzői

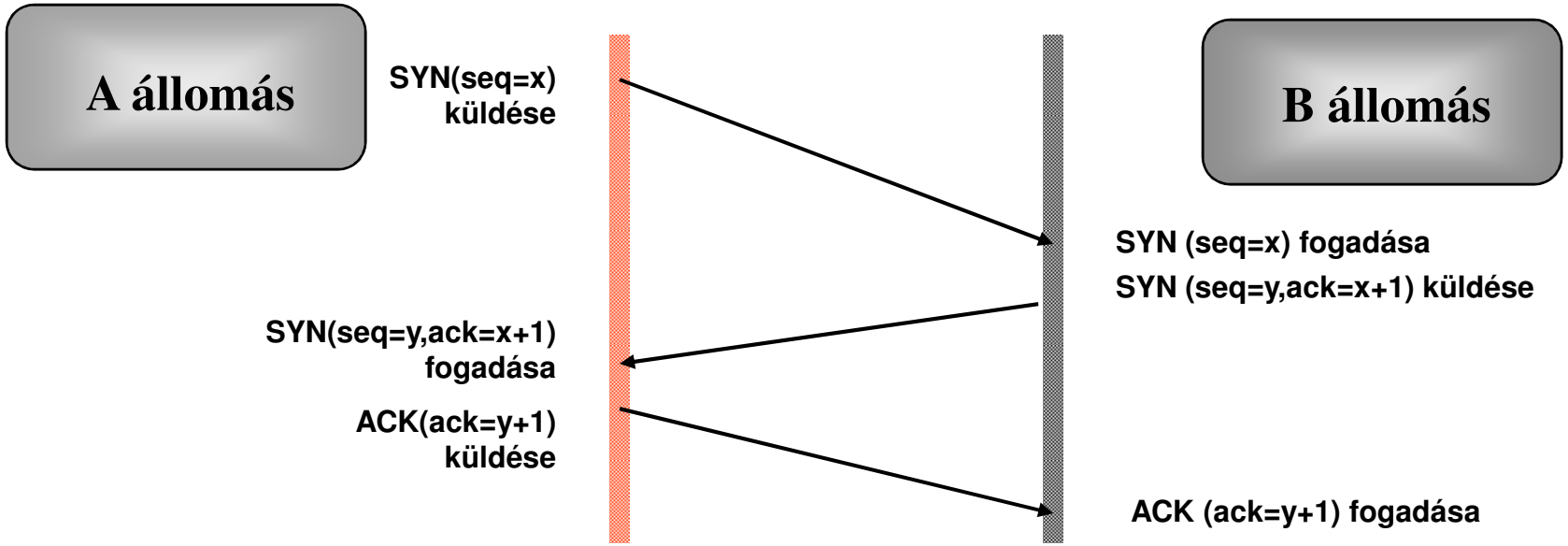
- **Újraküldés**
  - a TCP feladata, hogy adott esetben (pl. egy bizonyos idő lejártával) az egyes csomagokat újra elküldje, mivel lehet, hogy az előző példány elveszett valahol
- **Sorrendhelyes átvitel**
  - A célállomáson a megérkezett csomagok sorrendje nem biztos, hogy az elküldés sorrendjével megegyezik, ezért a TCP feladata ennek a rendezése is (ha szükséges)
- **Csomagduplázódás**
  - A TCP a csomagduplázás ellen is védelmet nyújt
- **Megbízhatóság**
  - az ún. PAR (Positive Acknowledgement with Retransmission) technikával biztosítja. Ez azt jelenti, hogy a célállomás TCP-t megvalósító szoftvere nyugtázza a csomag kézbesítését, miután a hálózati szinttől (az IP-től) megkapta.
- **Megbízhatóság és késleltetés**
  - A TCP esetében a megbízhatóság azt jelenti, hogy az elküldött csomagok biztosan megérkeznek, de az esetleges újraküldések miatti késleltetésre nincs garancia
  - Valós idejű szolgáltatások esetén ezért nem javasolt a TCP használata

TCP jellemzői

- **Kapcsolatorientált**
  - Kapcsolatkiépítés három-utas kézfogással (sorszám meghatározása)
- **Több kapcsolat**
  - Egy hoston egyszerre több TCP kapcsolat is élhet, és itt is, mint az UDP-nél, az egyes kapcsolatok külön-külön TCP-porton (TSAP-on) vannak
- **Full-duplex adatfolyam**
  - A TCP-kapcsolatok full-duplexek, vagyis kétirányúak, és az elküldött adatokat a TCP strukturálatlan byte- folyamnak tekinti.
  - MSS: maximálisszegmens méret (maximum segment size)
- **Forgalomszabályozás (flow control)**
  - A küldő nem terheli túl a fogadót
- **Torlódáskezelés (congestion control)**



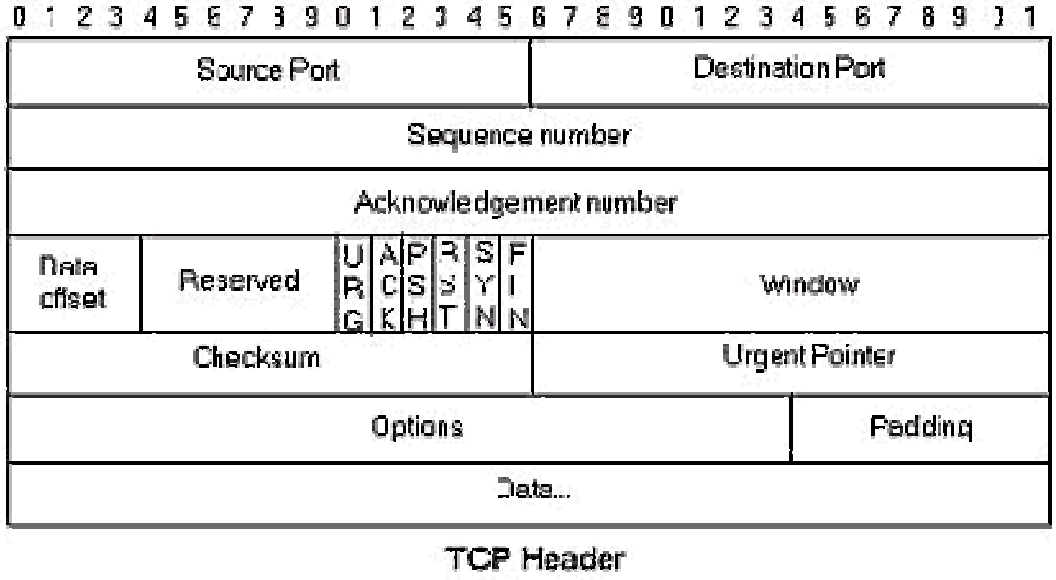
Háromfázisú kézfogás (3-way handshake)



SYN – szinkronjel, ACK – Nyugtázás

Az x az A, az y pedig a B állomás sorszáma

TCP fejléc



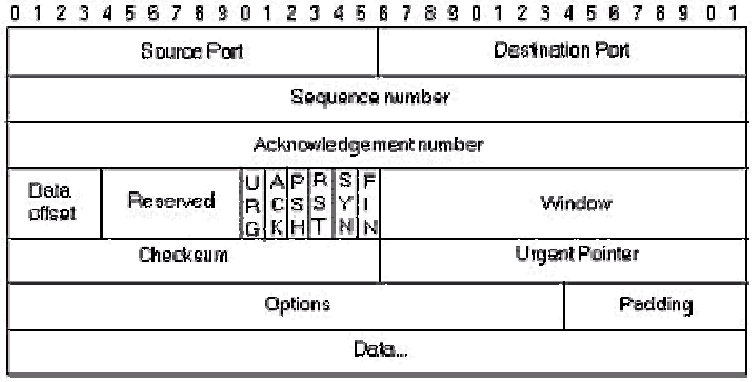
TCP Header

- Portszám (Source Port, Destination Port)

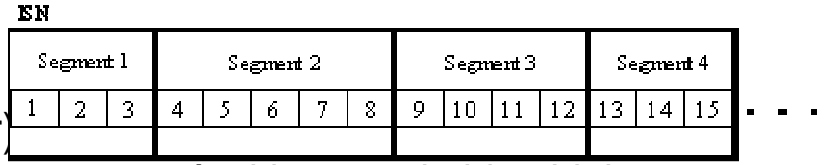
  A fontosabb, szélesebb körben használt protokollok egy "mindenki által ismert" sorszámú port-on várnak kapcsolatokra:
  - HTTP: 80
  - FTP: 20, 21
  - SSH: 22
  - SMTP: 25
  - Telnet: 32

TCP fejléc

- Sorszám (Sequence Number)
  - a vevő oldalt arról biztosítja, hogy minden adatot helyes sorrendben kapjon meg, és ne veszítsen el egyet se a datagrammok közül.
  - A TCP nem a datagrammokat, hanem az szegmenseket sorszámozza
  - 3-utas kézfogás



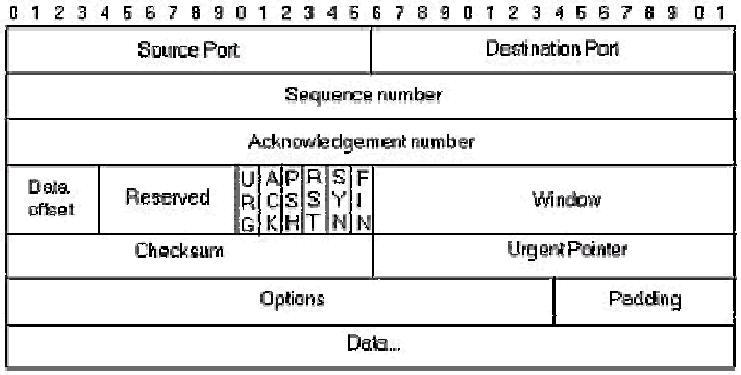
TCP Header



- Nyugta sorszám (Acknowledgement Number)
  - a rendeltetési helyre való megérkezést a vevő egy nyugtával hozza a küldő oldal tudomására
  - Például egy olyan csomag elküldése, amelynek nyugtamezőjében 1500 szerepel, azt jelenti, hogy az 1500-as oktetig bezárólag minden datagram eljutott a rendeltetési helyre

TCP fejléc

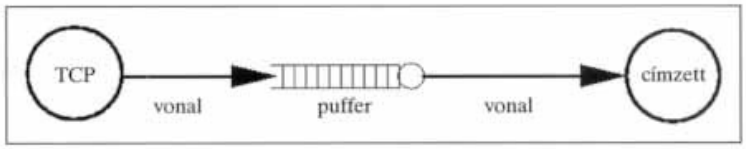
- Egybites változók
  - URG: sürgősségi mutató használatát engedélyezi
  - ACK: a nyugta érvényességét jelezi, 0 esetén a szegmens nem tartalmaz nyugtát, figyelmen kívül hagyható a mezeje
  - PSH: késedelem nélküli továbbítás kérése- pufferelés nélkül
  - RST: hoszt összeomlását vagy az összekötés helyreállításának igényét jelzi.
  - SYN: összekötés létesítésére irányul
    - kérés (CR): SYN=1 & ACK=0
    - elfogadás (CA): SYN=1 & ACK=1
  - FIN: összeköttetés bontását jelzi
    - a küldőknek nincs több továbbítani való adata

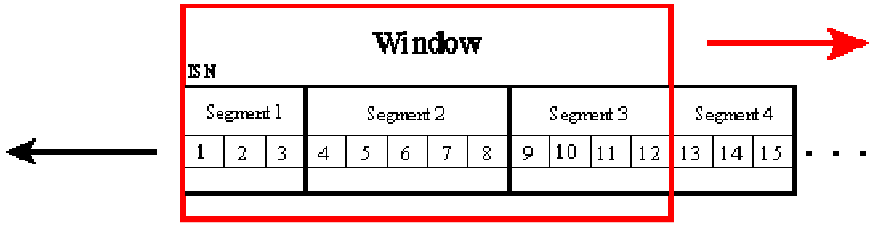


TCP Header

TCP forgalomszabályozás

- Cél, hogy a küldő ne terhelje túl a fogadót
  - A küldő nem akarja túltölteni a vevő-puffert azzal, hogy túl sokat, túl gyorsan küld
- Átviteli sebesség korlátjai
  - A vevő kapacitása
  - A hálózat kapacitása
- Adó oldali csomagok típusai
  - Elküldött – nyugtázott
  - Elküldött – még nem nyugtázott
  - Még nem elküldött – elküldhető
  - Még nem elküldött – még nem küldhető el
- Vevő oldali csomagok típusai
  - Megérkezett (nyugtázott)
  - Nem érkezett meg, de megérkezhet (képes fogadni)
  - Nem érkezhet meg (nem képes fogadni)
- A küldő ne árassza el a vevőt





TCP - Torlódásszabályozás

Torlódás:

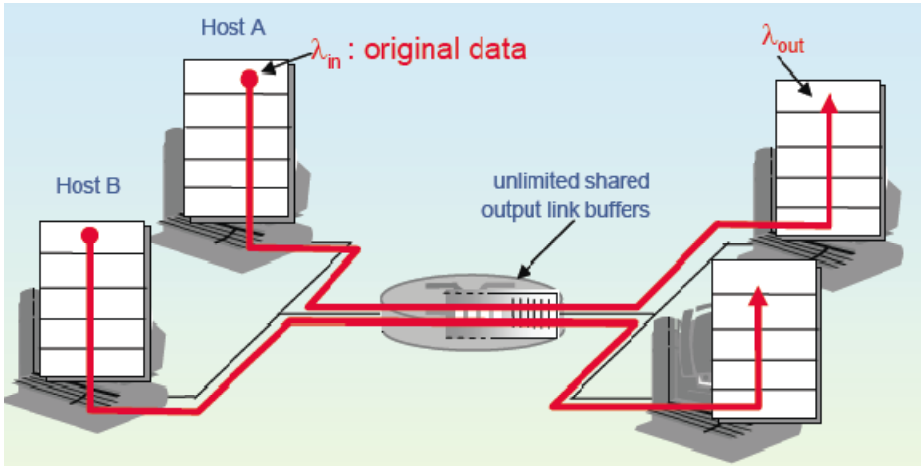
- Ha egyes hálózatrészek túltelítődnek akkor a csomagok mozgatása lehetetlenné válhat.
- A várakozási sorok, amelyeknek ezeket a csomagokat be kellene fogadniuk, állandóan tele vannak.
- A torlódás a csomaghálózatokban olyan állapot, amelyben a hálózat teljesítménye valamilyen módon lecsökken, mert a hálózatban az áthaladó csomagok száma túlságosan nagy.
- A teljesítménycsökkenés jelentkezhet oly módon hogy
  - a hálózat átbocsátóképessége (throughput) lecsökkent, anélkül, hogy a hálózat terhelését csökkentenénk
  - a hálózaton áthaladó csomagok késleltetése megnőtt.

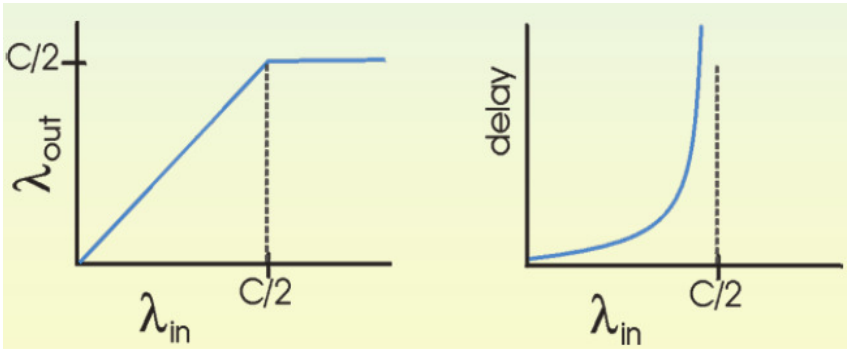
A torlódás okai és következményei

Példa

- Két küldő, két fogadó
- Egy router, végtelen puffer
- Nincs újraküldés

- Torlódáskor nagy késleltetés
- Maximális elérhetőátvitel





Torlódásvezérlési megközelítések

End-end torlódásvezérlés

- Nincs egyértelmű (explicit) visszacsatolás a hálózatból
- A torlódás a végberendezésben érzékelt veszteségben, késleltetésben jelenik meg
- Ezt a megközelítést használja a TCP

Hálózat által támogatott torlódásvezérlés

- A routerek nyújtanak visszacsatolást a végberendezéseknek
- Egyetlen bit jelzi a torlódást(SNA, DEC bit, TCP/IP ECN, ATM)
- Egyértelmű sebesség megadás a küldőnek, amellyel küldhet

TCP - Torlódásszabályozás

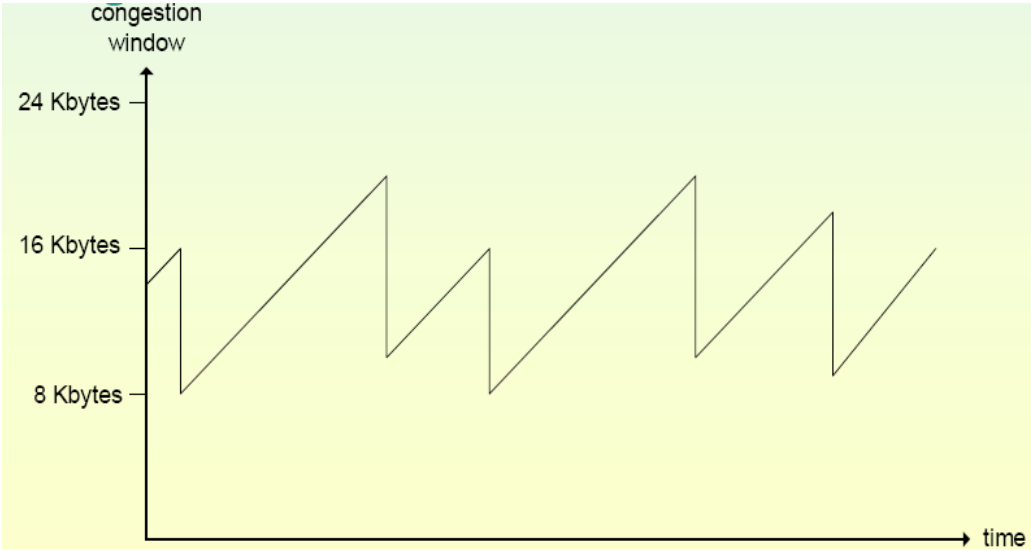
- Megközelítés:
  - addig növeljük az átviteli sebességet (ablakméretet), a használható sávszélesség kipróbálásával, amíg veszteség nem történik
- Csomagvesztés után a TCP megfelezi a hálózatba küldött csomagjainak számát
- Majd ismét növeli a küldési sebességét a következő ütközésig, vagyis csomagvesztésig
- Ennek megvalósítására az algoritmus használ egy torlódási ablak változót (congestion window - *cwnd*)

TCP - Torlódásszabályozás

- Csomagvesztés után a TCP óvatosabbá válik
  - Additívnövelés (AI)

    növeljük a *cwnd*-t 1MSS-sel minden RTT alatt, amíg csomagvesztést nem detektálunk
  - Multiplikatív csökkentés (MD)

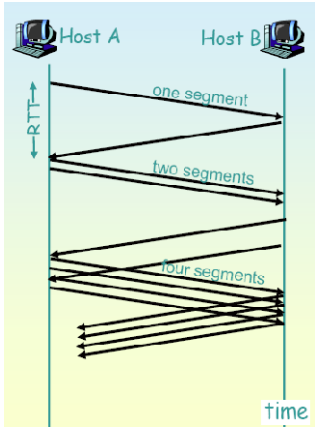
    csökkentsük a *cwnd*-t a felére csomagveszteség detektálásakor



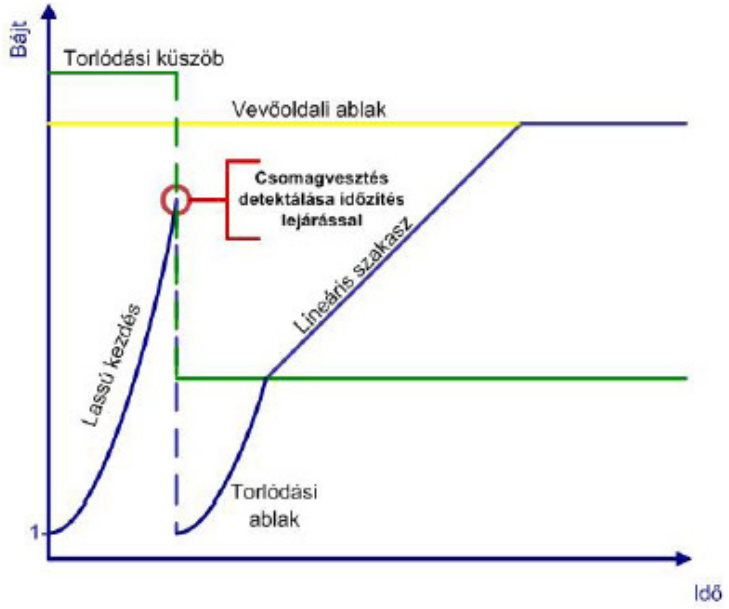
TCP - Torlódásszabályozás

Slow Start

- Amikor az összeköttetés létrejön, növeljük a sebességet exponenciálisan az első csomagvesztési eseményig
  - *cwnd* duplázása minden RTT-ben
  - *cwnd* növelése által minden kapott ACK-re



- Amikor a *cwnd* értéke elérte a timeout előtti értékének a felét (threshold) az exponenciális sebességnövelés helyett lineáris növelés



TCP variánsok

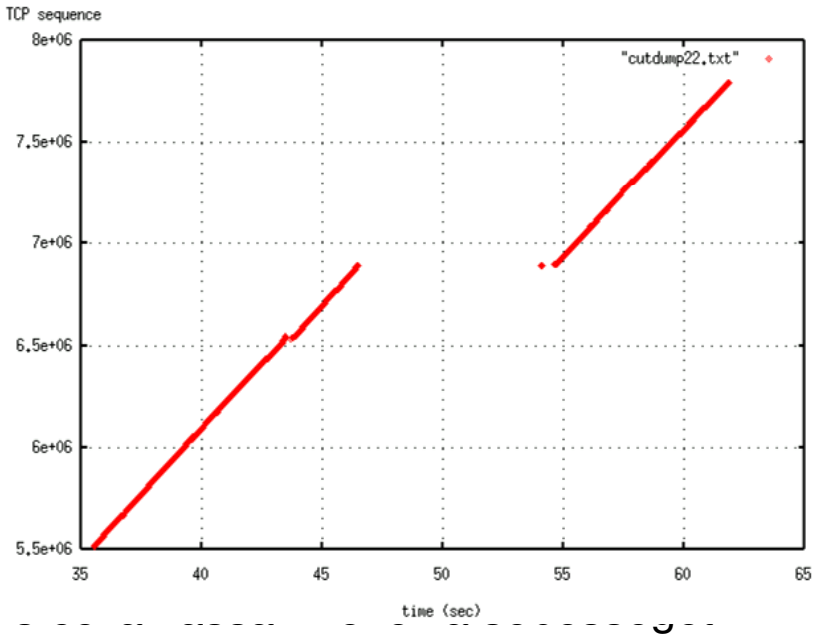
- Változtatás a protokoll torlódásszabályozási mechanizmusában
  - TCP Tahoe
  - TCP Reno
  - TCP New Reno
  - TCP SACK (Selective Acknowledgement)
  - TCP Vegas
  - TCP BIC (Binary Increase Congestion Control)
  - TCP CUBIC
  - TCP Westwood
  - TCP Hybla
  - Scalable TCP
  - HighSpeed TCP
  - H-TCP
  - TCP Veno
  - TCP-LP (Low Priority)
  - stb.

TCP vezeték nélküli környezetben

- A TCP nem képes különbséget tenni a csomag sérülése miatti csomagvesztés és a torlódás miatti vesztés között.
- Így minden csomagvesztés ugyanazt az adó részéről történő
  - torlódás elkerülési választ vonja maga után, ami az adó átküldési sebességének a csökkenését okozza
  - még akkor is, ha a hálózatban nincs torlódás
- A vezeték nélküli átvitel miatt bithibák jelentkeznek, amik csomagvesztésben nyilvánulnak meg.
- A TCP vezetékes környezetre optimalizált, így a csomagvesztést torlódásként értelmezi és csökkenti az átviteli sebességet.

TCP vezeték nélküli környezetben

- A TCP zajos csatornán indokolatlanul csökkentheti az adatátviteli sebességét
- Cellaváltás (az RTT hirtelen megnő), adatforgalom leáll



- Gyors → lass
- Lassú → gyo

TCP vezeték nélküli környezetben

- A TCP hibáinak kiküszöbölésére nehézkes, hiszen minden Internethez csatlakoztatott gépen van egy példány
  - Ezeknek együtt kell működniük
  - Lényegi változtatás így nem oldható meg
- A módosított TCP változatoknak kompatibilisnek kell maradniuk
- TCP vezeték nélküli környezetben:
  - Indirect TCP (I-TCP)
  - Snooping TCP
  - Mobile TCP (M-TCP)
  - Fast retransmit/fast recovery
  - Transmission/time-out freezing

TCP finomhangolás

- Finomhangolás a `sysctl` segítségével
- Torlódáskezelő algoritmus kiválasztása
  - Beállított érték

    ```
    net.ipv4.tcp_congestion_control = cubic
    ```
  - Melyeket választhatjuk? `net.ipv4.tcp_available_congestion_control = cubic reno`
  - Új algoritmusok beállítása

    A TCP torlódáskezelő algoritmusai modulként töltődnek be. A rendszer indításakor azonban nem minden ilyen modul töltődik be. Célja a memória megtakarítása.

    Az előző példában a *cubic* és *reno* már betöltött modulok, de sok más torlódáskezelő is automatikusan betöltődik, ha *sysctl* paranccsal be akarjuk állítani. Pl.

    ```
    sysctl -w net.ipv4.tcp_congestion_control=bic
    ```

TCP finomhangolás

Az elérhető modulok listája a `modprobe` parancs segítségével érhető el:

```
modprobe -l
```

Ez alapján már láthatjuk, hogy számos algoritmus elérhető

```
root@dccp-mcl:/home/dccp# modprobe -l | grep tcp

/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_htcp.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_probe.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_westwood.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_illinois.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_veno.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_highspeed.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_lp.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_scalable.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_yeah.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_bic.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_hybla.ko
/lib/modules/2.6.27-11-generic/kernel/net/ipv4/tcp_vegas.ko
```

TCP finomhangolás

- Vezetéknélküli estre használjunk *westwood*-ot, mert ennél az algoritmusnál csomagvesztés ≠ torlódás.
- TCP buffer size
  - A TCP buffer mérete visszafoghatja az átviteli sebességet, amennyiben nagy sebességű kapcsolatunk van, hiszen ez határozza meg, hogy hány csomagot küldhetünk ki egyszerre. A maximális buffer méret ezért korlátozhat

	min	default	max
`net.ipv4.tcp_wmem =`	`4096`	`16384`	`913920`
`net.ipv4.tcp_rmem =`	`4096`	`87380`	`913920`

  - Vigyázni kell arra, hogy a küldő ne árassza el a vevőt

Kliens/szerver alkalmazás

- **Hálózati alkalmazások *socket*ek használatával történik**
- ***Socket***
  - A *socket*ek végpontok, amelyekhez alulról (az operációs rendszer felől) az összeköttetések, míg felülről (a felhasználó felől) a folyamatok kapcsolódnak.
  - A *socket* rendszerhívás létrehoz egy *socket*et, egy operációs rendszeren belüli adatstruktúrát
  - a hívások paraméterei kijelölik a
    - címformátumot (pl. egy Internet nevet)
    - a socket típust (pl. összeköttetés-alapú vagy összeköttetés-mentes)
    - valamint a protokollt (pl.TCP, UDP, stb.)

TCP kliens/szerver alkalmazás

Hogyan kell TCP kliens-szerver alkalmazást írni C-ben

- Szerver
  - *Socket* létrehozása a `socket()` paranccsal
  - A *Socket Address* struktúra beállításai – cím/port beállítás
    - IP cím = INADDR_ANY
      bármely címről képes fogadni kapcsolatot
  - *Socket* és *Socket Adress* struktúra összekapcsolása
    `bind()`
  - Fogadásra kész *Socket*
    `listen()`
  - Fogadási ciklus
    `accept()` – bejövő kapcsolatok fogadása

TCP szerver

```
/*  Create the listening socket  */

   if ( (list_s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0
        ) {
        fprintf(stderr, "SERVER: Error creating listening socket.\n");
        exit(EXIT_FAILURE);
   }

   /*  Set all bytes in socket address structure to zero, and fill in the
        relevant data members   */

   memset(&servaddr, 0, sizeof(servaddr));
   servaddr.sin_family      = AF_INET;
   servaddr.sin_addr.s_addr = htonl(INADDR_ANY);
   servaddr.sin_port        = htons(port);

   /*  Bind our socket addresss to the listening socket, and call listen()  */

   if ( bind(list_s, (struct sockaddr *) &servaddr, sizeof(servaddr)) < 0 ) {
        fprintf(stderr, "ECHOSERV: Error calling bind()\n");
        exit(EXIT_FAILURE);
   }

   if ( listen(list_s, LISTENQ) < 0 ) {
        fprintf(stderr, "ECHOSERV: Error calling listen()\n");
        exit(EXIT_FAILURE);
   }
```

```
/* Enter an infinite loop to respond to client requests and echo input */

   while ( 1 ) {

        /*  Wait for a connection, then accept() it  */

        if ( (conn_s = accept(list_s, NULL, NULL) ) < 0 ) {
           fprintf(stderr, "ECHOSERV: Error calling accept()\n");
           exit(EXIT_FAILURE);
        }


        /*  Retrieve an input line from the connected socket then simply
        write it back to the same socket.     */

        Readline(conn_s, buffer, MAX_LINE-1);
        Writeline(conn_s, buffer, strlen(buffer));

        /*  Close the connected socket  */

        if ( close(conn_s) < 0 ) {
           fprintf(stderr, "ECHOSERV: Error calling close()\n");
           exit(EXIT_FAILURE);
        }
   }
}
```

TCP kliens/szerver alkalmazás

- Kliens
  - *Socket* létrehozása a `socket()` paranccsal
  - A *Socket Address* struktúra beállításai – cím/port beállítás
    - IP cím = AF_INET

      a kliens kapcsolódási címe
  - Kapcsolódás a szerverhez

    `connect()`
  - Kommunikáció a szerverrel

    `write(),read()`

TCP kliens

```
/*  Create the listening socket  */

   if ( (conn_s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0 ) {
         fprintf(stderr, "ECHOCLNT: Error creating listening socket.\n");
         exit(EXIT_FAILURE);
   }


   /*  Set all bytes in socket address structure to
       zero, and fill in the relevant data members   */

   memset(&servaddr, 0, sizeof(servaddr));
   servaddr.sin_family      = AF_INET;
   servaddr.sin_port        = htons(port);


   /*  Set the remote IP address  */

   if ( inet_aton(szAddress, &servaddr.sin_addr) <= 0 ) {
         printf("ECHOCLNT: Invalid remote IP address.\n");
         exit(EXIT_FAILURE);
   }


   /*  connect() to the remote echo server  */

   if ( connect(conn_s, (struct sockaddr *) &servaddr, sizeof(servaddr) ) < 0 ) {
         printf("ECHOCLNT: Error calling connect()\n");
         exit(EXIT_FAILURE);
   }
```

```
/*  Get string to echo from user  */

   printf("Enter the string to echo: ");
   fgets(buffer, MAX_LINE, stdin);


   /*  Send string to echo server, and retrieve response  */

   Writeline(conn_s, buffer, strlen(buffer));
   Readline(conn_s, buffer, MAX_LINE-1);


   /*  Output echoed string  */

   printf("Echo response: %s\n", buffer);

   return EXIT_SUCCESS;
}
```

Socket - csomagküldés

```
/*  Read a line from a socket  */

ssize_t Readline(int sockd, void *vptr, size_t maxlen) {
    ssize_t n, rc;
    char    c, *buffer;

    buffer = vptr;

    for ( n = 1; n < maxlen; n++ ) {

        if ( (rc = read(sockd, &c, 1)) == 1 ) {
            *buffer++ = c;
            if ( c == '\n' )
                    break;
        }
        else if ( rc == 0 ) {
            if ( n == 1 )
                    return 0;
            else
                    break;
        }
        else {
            if ( errno == EINTR )
                    continue;
            return -1;
        }
    }

    *buffer = 0;
    return n;
}
```

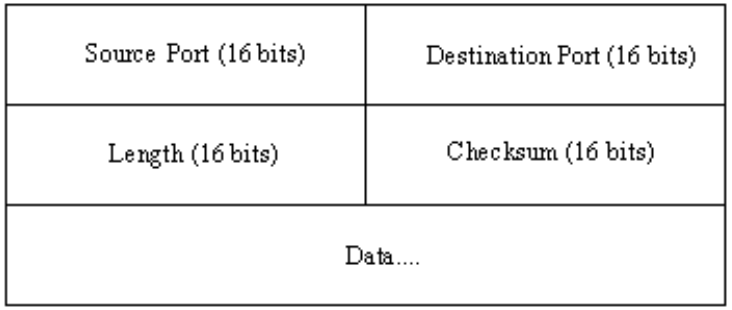
```
/*  Write a line to a socket  */

ssize_t Writeline(int sockd, const void *vptr, size_t n) {
    size_t      nleft;
    ssize_t     nwritten;
    const char *buffer;

    buffer = vptr;
    nleft  = n;

    while ( nleft > 0 ) {
        if ( (nwritten = write(sockd, buffer, nleft)) <= 0 ) {
            if ( errno == EINTR )
                    nwritten = 0;
            else
                    return -1;
        }
        nleft  -= nwritten;
        buffer += nwritten;
    }

    return n;
}
```

UDP

- User Datagram Protocol [RFC-768]
- 1980
- Az UDP sokkal gyorsabb protokoll, mint a TCP protokoll
- Nem megbízható adatátvitel
- Multimédiás alkalmazások esetén jól alkalmazható, ahol a késleltetés a kritikus
- A TCP-vel ellentétben nem ellenőrzi az adatok sértetlen átvitelét
  - ezért nem képes az elveszett vagy sérült csomagok pótlására
- Ezen kívül a fogadás sorrendjét sem garantálja a vételi oldalon.

UDP/IP fejléc

- Source Port
  - A forrásportot azonosítja
  - Válaszolni erre a portra lehet
- Destination Port
  - Célportot azonosítja
- Length
  - A datagram mérete bájtokban
  - A fejléc és a felhasználói adat együtt
- Checksum
  - 16-bit ellenőrzőösszeg
  - A fejléc és a felhasználói adatokra együtt számolandó



- UDP-Lite
  - Részleges ellenőrzőösszeg alkalmazása (partial checksum)

UDP hangolás

- Hasonlóan a TCP-hez, az UDP esetén is módosítható a bufferméret.

```
root@dccp-mcl:/home/dccp# sysctl -a | grep udp

net.ipv4.udp_mem = 23814        31752   47628
net.ipv4.udp_rmem_min = 4096
net.ipv4.udp_wmem_min = 4096
```

- Túl kicsi bufferméret esetén túlcsordulás történhet, ami csomagvesztéshez vezet
- Túl nagy bufferméret esetén
  - Lassabb feldolgozás várható és az adatok elveszthetik aktualitásukat.
    - Ha pl. 10 csomagonként van friss adat, akkor a bufferméretet ennek megfelelően érdemes beállítani.
  - Jelentős memória használat, ha több alkalmazást is futtatunk egyszerre

UDP hangolás

- Késleltetés vs. bufferméret
  - Feltéve, hogy ismerjük az átlagos küldési sebességet a késleltetés számolható:

    Max Latency = Buffer Size / Average Rate

http://www.29west.com/docs/THPM/udp-buffer-sizing.html

UDP kliens/szerver alkalmazás

- A módszer hasonlatos a TCP esethez
- A *Socket* létrehozásánál az UDP protokollt kell megadnunk
- TCP:

```
socket(AF_INET, SOCK_STREAM, IPPROTO_TCP)
```

- UDP:

```
socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)
```

- Automatikus protokollválasztás:

```
socket(AF_INET, SOCK_DGRAM, 0)        → UDP
socket(AF_INET, SOCK_STREAM, 0)       → TCP
```

Összefoglalás

- ISO/OSI és TCP/IP architektúra
- Szállítási réteg
- Alkalmazások típusai
- Vezeték nélküli hálózatok
  - Tulajdonságai
  - Mobilitás támogatás
- *Sysctl* hangolás
- Protokollok
  - TCP
    - Tulajdonságok
    - Torlódáskezelés
    - TCP vezeték nélküli környezetben
    - TCP beállítások
  - UDP

Kérdések?

KÖSZÖNÖM A FIGYELMET!

Híradástechnikai Tanszék

Szabó Sándor
Huszák Árpád
BME Híradástechnikai Tanszék
szabos@hit.bme.hu