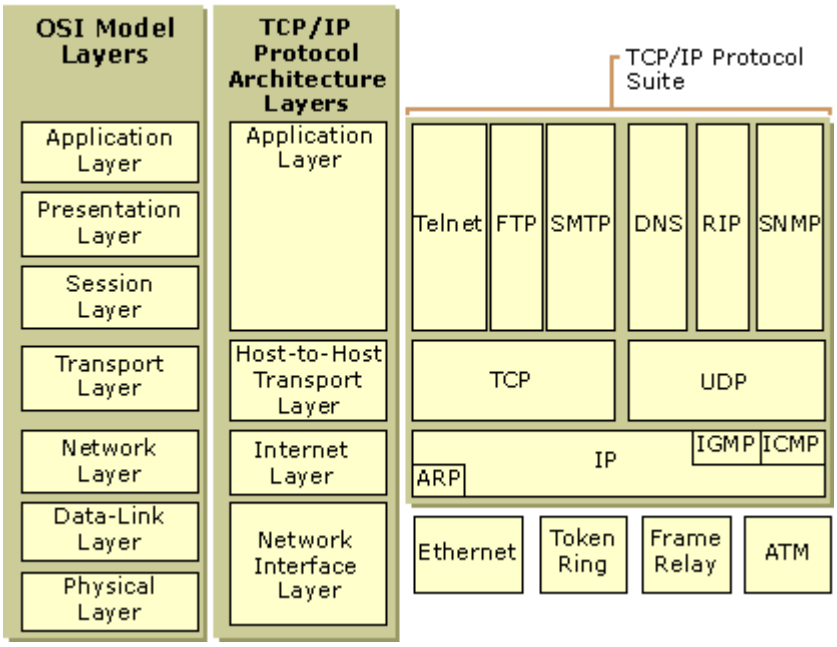
TRANSMISSION CONTROL PROTOCOL (TCP) bevezetés[1]



Az áttekintő térkép eligazított minket arról, hogy hol járunk, majd nézzük meg külön az aktuális részeket:

Alkalmazás I/A	**Alkalmazás II/A**	**Alkalmazás III/A**	*A transzport réteg kliensei*	**Alkalmazás I/B**	**Alkalmazás II/B**	**Alkalmazás III/B**
Transzport réteg (TCP-UDP)			*Protokoll stack*	***Transzport réteg (TCP-UDP)***		
Hálózati réteg (IP)				Hálózati réteg (IP)		
~~				~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~		
Fizikai réteg				Fizikai réteg		

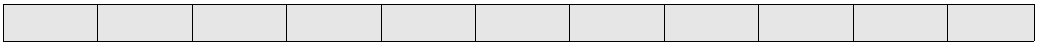
A transzport réteg feladata, hogy kiszolgálja az alkalmazások adatátviteli igényeit. (Legalábbis most fogadjuk el, hogy az alkalmazásokkal közvetlen kapcsolatban áll.) Az LLC rétegnél már találkoztunk olyan problémával, hogy az alkalmazások (azaz a kliensei) közül kellett választania. A transzport réteg protokolljának is kell ilyen szolgáltatást biztosítania. A transzport réteg (protokollja) az ún. **portszám** alapján különbözteti meg a klienseit. Az LLC rétegben a különböző SAP-ok kapcsán már találkoztunk ezzel a módszerrel. A Transzport réteg szolgáltatás elérési pontjainak TSAP a hívatalos neve. (Pontosabban egy TSAP-ot az IP cím és a TCP vagy UDP portszám együtt alkotnak. A TSAP egy OSI kifejezés, a gyakorlatban inkább a SOCKET-ként terjedt el.)

Láttuk, hogy az IP nem garantálja a csomagok megérkezését és sorrendjét, bár a továbbítás során legjobb tudása szerint jár el. A TCP viszont megbízható kapcsolatot létesít két (!) végpont között és ezen kapcsolat létrejötte után tetszőleges hosszúságú üzeneteket kaphat a kliens alkalmazástól, illetve tetszőleges hosszúságú üzeneteket adhat át nekik. A hálózaton történő sikeres továbbításhoz ezeket az üzeneteket „rövid darabokra” (szegmensekre) kell

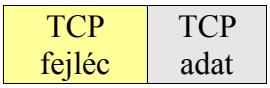
1 Az Internet elődjén az ARPANET-en a szállítási protokoll az Network Control Protocol (NCP) volt. A TCP „szabvány” leírása megközelítően 100 oldalnyi terjedelmű. Mi csak áttekinthető jelleggel ismertetjük a protokollt, remélve azt, hogy a használathoz ez is elég. Aki TCP szubrutin csomagot szeretne, pl. egy mikrovezérlőhöz, annak javaslom az RFC 793 tanulmányozását vagy egy kész rutinkönyvtár letöltését a Netről.

tördelni. Gondoljunk arra, hogy pl. egy ETHERNET LAN-ban max. 1500 bájtos datagramokat tudunk továbbítani. Ha a példa kedvéért feltételezzük, hogy az alkalmazói rétegben egy FTP alkalmazás segítségével MByte méretű állományt viszünk át egyik gépről a másikra, akkor belátható, hogy jó néhány datagramot kell kialakítani az FTP kapcsolat alatt. A TCP egy fejlécet illeszt az aktuálisan átküldendő adatok elé, így jön létre egy TCP szegmens. A TCP szegmenst az IP továbbítja IP datagram(ok)ként. Az IP datagramot pedig a MAC protokoll keretezi be, hogy továbbíthassa fizikailag. Vétel esetén egy fordított a folyamatban a protokoll verem minden szintje "lehámozza" (**decapsulation)** saját fejlécét a datagramról. Az üzenet egy egységet alkot, de már most megjelöltük leendő szegmenseit:

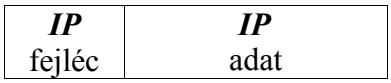
Üzenet



TCP szegmens (TPDU[2]):

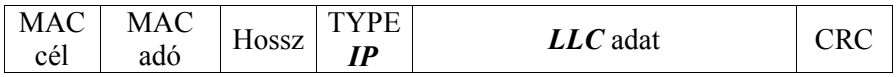


IP datagram (NPDU):



Encapsulation-Decapsulation

MAC keret:



Figyeljük meg, hogy az adatkapcsolati rétegben **keret**, a hálózati rétegben **csomag** és a transzport rétegben pedig **szegmens** névvel azonosítják az adott réteg által kezelt adategységet (információs strukturát). Nem látható, de megemlítjük, hogy a fizikai rétegben **bitsorozat**, míg a viszony rétegben **tranzakció** a stuktura neve.

A TCP protokollról sok mindent megtudhatunk, ha a fejlécét részletesebben megvizsgáljuk:

Source port (16 bit)			Destination Port (16 bit)
SEQNo (32 Bit)			
ACKNo (32 Bit)			
HLEN (4 bit)	Res (6 bit)	CODE Bit (6 bit)	Window (16 bit)
Ellenőrző összeg (16 bit)			Urgent Pointer (16 bit)
OPTION (n * 32 bit)			
Alkalmazói réteg adatai (0 vagy páros számú byte)			

A **cél port** és **adó port** azonosítja (mintegy szállítási rétegbeli cím) a TCP szolgáltatásait használó klienseket[3]. A 1024 alatti portszámok az ún. jól ismert portszámok, amelyeket általában a standard hálózati alkalmazások szerver komponenséhez rendeltek (http://www.iana.org/assignments/port-numbers), pl.:

FTP	20 (data), 21 (control)	Bootp client	68
Telnet	23	Gopher	70
SMTP	25	Finger	79
HTTP	80 (web server)	POP3	110
Who Is	43	NetBIOS Name server[4]	137
Kerberos	88	NetBIOS Datagram[4]	138
POP2	109	NetBIOS Session[4]	139

Az 1024 alatti portszámok sajátossága, hogy több OS, pl. UNIX alatt csak különleges

2 A jegyzetben kétféle TPDU-val fogunk találkozni: TCP ill. UDP szegmensekkel

3 A transzport protokoll számára a magasabb rétegben implementált szolgáltatások klienseknek tekinthetők függetlenül attól, hogy azok a saját szintjükön kliensek vagy szerverek, pl. a HTTP szerver és kliens egyaránt a TCP kliense.

4 Csak akkor, ha a NetBIOS TCP/IP fölött megy, ugyanis NETBEUI fölött a NetBIOS nem használ TCP portot.

jogokkal rendelkező programoknak engedélyezett a használatuk, így a felhasználó számára biztonságot jelent az, hogy egy rendszergazda vagy a rendszer által indított programmal, nem pedig egy "feketén" elindított bizonytalan célú programmal kerül kapcsolatba. 1024 fölött is léteznek jól ismert portszámok, pl.: HTTP=8080, de ilyen portszámot bárki nyithat.

A portok 3 kategóriába sorolhatók:

- **Jól ismert (well known) portszámokat** (0..1023) használó alkalmazások számos OS-től különleges támogatást kapnak. (Említettük, hogy általában az alkalmazások kiszolgáló komponensei kapják ezeket a portszámokat.)
- **Regisztrált portszámok**: 1024..49151. A 49152 alatti portszámokat használó alkalmazások készítőjének az adott portszám használatát regisztráltatni kell az Internet Assigned Numbers Authority (IANA) társaságnál.
- **Dinamikus és/vagy privát portok** 49152..65535. (Pl. az FTP kliens adatátvitelhez innen választ.)

A gépünkön a **c:\windows\…\etc\services** állomány tartalmazza a szolgáltatások és portszámok összerendelését. Az egyes szolgáltatások a registry-ben letilthatók.

Az alkalmazási rétegben szintén protokollok találhatók, amelyek különböző célra lettek kifejlesztve: pl. a fájl átvitelhez a File Transfer Protocol (FTP), vagy másik példaként a levelezéshez használható Simple Mail Transfer Protocol (SMTP) és a szintén levelezési célú Post Office Protocol (POP). Az ábrán az alkalmazási rétegben futó FTP, a TCP/IP segítségével adatokat cserél a két gép között.

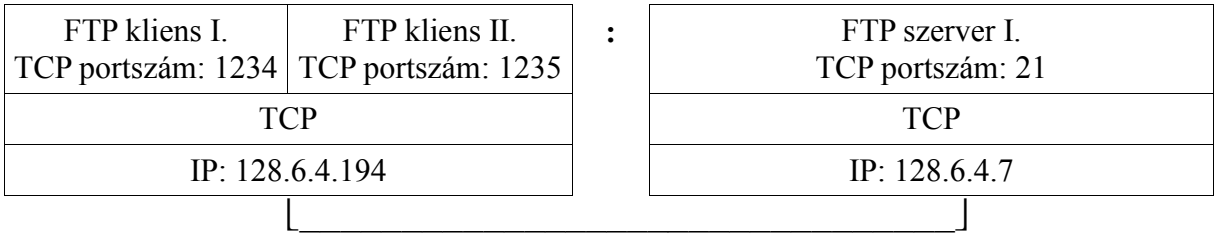
Összefoglalva az eddigieket: a két gép (vagy alkalmazás) kapcsolatában több protokoll kap szerepet. Először is az általános adatátvitelt biztosító, ún. low-level protokollok (pl. TCP, UDP, IP, ARP, stb.), végül a magasabb alkalmazási szintűek (pl. FTP, SMTP, POP, stb.) Az IP cím kijelöli, hogy mely gépek vesznek részt a kapcsolatban. A TCP portszám az előbb kiválasztott két gépen futó egyes alkalmazási protokollokat azonosítja. A TCP ennek alapján tudja a beérkezett szegmenseket a megfelelő felhasználóhoz eljuttatni. Egy adott gépen futó többféle alkalmazási protokoll számára különálló logikai csatornákat képes biztosítani. Másképp megfogalmazva a TCP, az alkalmazási rétegben implementált konkurens kliensei számára (a portok ismeretében) egy-egy socket-et (a kommunikációs végpont csatlakozóját) kell létrehozni az adott TCP kliens aktivizálásakor. A socket-eket tehát az IP és portszám különbözteti meg. A portszámok között vannak olyanok, amelyek a standard alkalmazási protokollokhoz tartoznak (ezek többnyire a fenti táblázatban található 1024 alatti jól ismert portszámok), illetve vannak olyanok, amelyeket dinamikusan allokál magának egy-egy alkalmazási protokoll, vagy alkalmazás.

Példák a portszámok használatára:

Az **első példában** az FTP alkalmazói szintű protokollt vizsgáljuk meg. Erről tudnunk kell, hogy az alkalmazói szintű protokolloknak van szerver, ill. kliens oldali programja. Először a szervert indítjuk el, amely a jól ismert portszámok közül a 21-et foglalja le (a kliens-szerver jellegű felhasználói programok közül általában a szerver használja a jól ismert portokat), majd várakozik a kliens kapcsolódási kérelmére. Valahol egy másik gépen elindítjuk az FTP klienst, amely választ egy szabad portszámot magának, pl. 1234. A kliens, a cél portszámaként 21-et ad meg (a szerverének jól ismert portszámát). Előfordulhat, hogy az FTP klienst futtató gépen elindítják a kliens második ''példányát'' úgy, hogy a cél gép, sőt azon a cél protokoll ugyanaz, mint az előbbi esetben. Az FTP kliens második példánya egy új portszámot foglal magának, amely a példa kedvéért legyen 1235. Ekkor a címeket tekintve kétféle IP datagram halad majd a hálózaton[5] az FTP szerver felé:

5 Túlzás, hogy a hálózaton, mert minden egy gépen belül zajlik ebben a példában.

TCP és UDP



1. **Adó IP: 128.6.4.224 Adó port: 1234 Cél IP: 128.6.4.5 Cél port 21**

2. **Adó IP: 128.6.4.224 Adó port: 1235 Cél IP: 128.6.4.5 Cél port 21**

Az IP címből látszik, hogy a TCP ugyanazon gépek között alakít ki logikai 'csatornát' az IP segítségével. Pontosabban a TCP két különböző logikai csatornát hozott létre, mert ha már az egyik portszám eltérő a „port párokban" (21-1234, 21-1235) akkor az már külön csatornának számít. Mindegyik csatornán belül kialakul két külön irányú (szimplex) összeköttetés, tehát végeredményben egy-egy logikai csatorna duplex.

Saját gépünkön (is) megvizsgálhatjuk az alkalmazások, szolgáltatások (esetleg jó vagy rosszindulatú démonok, trójai programok) által megnyitott portokat, pl. a **portqry.exe** programmal (http://support.microsoft.com/?id=832919). A program a gépen futó processzek által használt portokat listázza ki. A példában a **skype** Internetes telefon program portjait mutatja:

```
Process ID: 3716 (Skype.exe)

Process doesn't appear to be a service

PID     Port            Local IP        State           Remote IP:Port
3716    TCP 80          0.0.0.0         LISTENING       0.0.0.0:2064
3716    TCP 443         0.0.0.0         LISTENING       0.0.0.0:2128
3716    TCP 16975       0.0.0.0         LISTENING       0.0.0.0:2288
3716    TCP 4403        192.168.0.112   ESTABLISHED     89.139.212.170:21733
3716    UDP 443         0.0.0.0                         *:*
3716    UDP 16975       0.0.0.0                         *:*
3716    UDP 1061        127.0.0.1                       *:*
```

Az alkalmazások közötti kapcsolat nem öncélú, hanem valamilyen erőforrás (objektum) elérésére irányul. Az erőforrás lehet egy levél, egy letöltendő program, egy weblap, stb. Ezek az erőforrások fizikailag valamilyen állományhoz kötődnek, pl. egy weblap egy html állományhoz. Az alkalmazási rétegben az **URL (Uniform Resource Locator)** formátum használatos az elérendő erőforrás megnevezéséhez. Az URL alakja **protokoll://hosztnév/útvonal/fájlnév** ill. ha nem szabványos portszámon érhető el az alkalmazás szerver, akkor **protokoll://hosztnév:portszám/útvonal/fájlnév**

Az URL-t az alsóbb rétegek számára érthetővé kell tenni, ezért a hosztnév IP címmé kerül feloldásra, a protokollnak pedig a portszám megfelelőjét használják. A fájl az erőforrás, amit az adott protokoll segítségével feldolgoz a rendszer.

Példák:

http://members.chello.hu/blathy
http://members.chello.hu/blathy/index.html
http://195.34.133.121/blathy/index.html

Az iskolánk weblapjának elérését láthattuk a példákban. A **members.chello.hu** nevű gépen a **http** protokoll szerint dolgozó kiszolgáló programmal szeretnénk a kapcsolatot felvenni. (A gép neve helyett az IP címe is használható). A **http** megadása egyrészt utalás az Internet egyik névterére és egyben a TCP 80-as portra hivatkozást is jelent. Az előzőekben meghatározott kiszolgáló programnak megadjuk, hogy a **/blathy** könyvtárban találja azt a fájlt (erőforrást) amelyik minket érdekel. Egy http kiszolgáló (a megadott könyvtáron belül) alapbeállítás szerint az **index.html** fájl tartalmát szolgáltatja nekünk, ezért az index.html megadása el is hagyható.

Néha előfordul, hogy a szerver alkalmazást nem a szabványos portszámra konfigurálják, mert pl. egy gépen több (pl. http) szerver van. Ilyenkor külön megadjuk az URL-ben, hogy az erőforrást szolgáltató alkalmazás melyik porton érhető el. A http kiszolgálóknál néha előfordul, hogy a 8080 portszámot használják. Ez „rizikós”, mert a felhasználónak ezt az eltérő portszámot ismernie kell:

http://195.34.133.121:8080/blathy/index.html
http://members.chello.hu:8080/blathy/index.html

Az URL a tartalmazhatja az erőforrás eléréséhez szükséges felhasználó azonosítót (user name), ill. a jelszót is. Részletesebb leírások URL-je: http://www.w3.org/Addressing/ ill. http://www.cis.ohio-state.edu/cgi-bin/rfc/rfc1738.html

Térjünk vissza a TCP-hez

Az FTP alkalmazások működés közben egy érdekes jelenség is megfigyelhető a TCP kapcsolatban. A működésének különböző fázisaiban újabb portot foglal le az FTP szerver, mert más port tartozik a parancsok és más az adatok továbbításához (21 illetve 20-as portok), természetesen ezt az FTP kliens is “tudja”. (Egy FTP szerver-kliens kapcsolat tehát két-két TCP logikai vonalat használ. Az FTP szerver 21-es portját a parancsokhoz és a szerver 20-as portját az adatokhoz, míg a kliens által lefoglalt portok sorszáma futás közben derül ki.) Itt említjük meg, hogy a HTTP–re is jellemző több port használata.

A **második példában** az SMTP és a portszámok kapcsolatát vizsgáljuk. Az SMTP-hez szintén egy kliens-szerver alkalmazás tartozik. Az SMTP szerver a 25-ös portot figyeli. Egy levelező programot elindítva megírhatunk egy levelet. A levél elküldéséhez a levelező programunk egy TCP kapcsolatot alakít ki a levél címzettjének SMTP szerverével. A levelező program (SMTP kliens) egy 1023 fölötti portszámot választ magának, míg a címzett gépen futó SMTP szerver a jól ismert 25-ös portszámot használja. A TCP kapcsolatot egy levél továbbítás során tehát a fenti két portszám jellemzi. Természetesen az SMTP kliens és szerver célként mindig a másik IP és portszámát használja:

Szerver	Kliens	Adó_IP + 25-ös port – Vevő_IP + 1024-es port
Szerver	Kliens	Adó_IP + 1024-es port – Vevő_IP + 25-ös port

A portok után nézzük a TCP fejléc további mezőit! A megértésük könnyebb, ha a szöveges magyarázatot követő ábrákat is tanulmányozzuk!

SEQNo (sorszám) és **ACKNo** (nyugtaszám). (Sequence Number és Acknowledgement Number.) Láttuk, hogy a TCP-TCP kapcsolatban az üzeneteket (tehát a logikailag egybe tartozó információkat, pl. az FTP esetén egy fájlt) szegmensekre tördeli a TCP program az átviteli utakra jellemző MTU miatt. A szegmensek jellemzőiről a TCP–TCP kapcsolat kézfogási folyamatában állapodnak meg. Ebben a valamiért **three-way handshake** nevezetű kézfogásban (kliens és szerver közötti szinkronizációban) a következő események történnek:

TCP és UDP

Sorszám:	Ügyfél által küldött szegmens tartalma és (**típusa**):	Kiszolgáló által küldött szegmens tartalma és (**típusa**):
1.	SYN=1 és ACK=0 mellett elküldi a portszámát és a kezdeti sorszámát (SEQNo) **(Connection Request TPDU)**	-
2.	-	SYN=1 és ACK=1 mellett elküldi a saját kezdeti sorszámát (SEQNo), de nyugtázás képpen a kliens kezdeti sorszámának is 1-el növeli az értékét (ACKNo) **(Connection Accepted TPDU)**
3.	ACK=1 mellett elküldi nyugtázás képpen a szerver kezdeti sorszámát 1-el növelve (ACKNo). **(Adat TPDU)**	-

A szegmens méretét az adott kapcsolat által érintett útvonalra jellemző MTU határozza meg. (Láttuk, hogy az IP képes felderíteni egy adott útvonal MTU értékét (path MTU).) Emlékeztetőként megemlítjük, hogy pl. egy olyan LAN-on, amely ETHERNET II típusú MAC kereteket használ 1500 bájtba kell beférnie egy TCP és IP fejléccel ellátott datagramának. Az Interneten javasolt a max. 536 bájtos TCP adatmező. Ez 556 bájtos lesz a TCP fejléccel kiegészítve és 576 bájtos, ha IP fejlécet is kap. Az 576 bájtos IP csomag illeszkedik az Internet gerincét található X.25-ös összeköttetések MTU értékéhez.

Más szempontok is befolyásolhatják a szegmensek méretét, pl. az, hogy az adott TCP–TCP kapcsolatot realizáló csatornán milyen arányban sérülnek meg az datagramák.

A példa kedvéért feltételezzük, hogy 1000 bájtnyi adatot tartalmaz egy-egy küldött TCP szegmens[6]. Akkor az első TCP szegmensben a SEQNo 0[7], a másodikban 1000 a harmadikban 2000. Tehát azt láthatjuk itt, hogy az üzenet hányadik bájtjától kezdődnek, az aktuális szegmensben lévő adatok. A vevő ennek segítségével tudja az eredeti üzenetet helyreállítani. Másképpen megfogalmazva, ha FTP-vel továbbítunk egy 67500 byte-os állományt, akkor ideális esetben 68 db TCP szegmenst kell elküldeni a társ TCP felé. A társ TCP minden beérkezett **bájtot** nyugtázni fog[8] a küldő TCP felé. A nyugtázó ACK szegmens TCP fejrész alakjában érkezik meg, adatok nélkül. Az ACKNo mező mutatja, hogy az üzenet hányadik bájtját kéri a vevő TCP rutin. Az előbbi példánál maradva először az üzenet 0, majd az 1000. majd a 2000. bájtját kéri a TCP. Az előbbi nyugták úgy is értelmezhetők, hogy 999.-ig, 1999.ig, stb. minden üzenet bájt hibátlanul megérkezett a vevő TCP-hez. Nagyon hosszú üzenetek esetén elképzelhető, hogy a sorszám (SEQNo) mező túlcsordul és előről kezdődik a sorszámozása az adott üzenet bájtjainak. Ez nem probléma, mert azok a TCP szegmensek amelyek már megkapták az adott sorszám értékét (pl. a 0-át és 1000-et) már régen "kimúltak" a hálózaton. A túlcsordulás ugyanis még egy nagyon gyors hálózaton is perc nagyságrendű idő alatt következik be.

Összefoglalás: a TCP a megbízható adatátvitel érdekében pozitív[9] nyugtázást és szükség esetén újraküldést alkalmaz.

Window: Említettük az előbb, hogy a TCP minden szegmenst nyugtáz. Ez egy kevés hibát produkáló átviteli csatornán felesleges lehet, mert a nyugtázó (szerviz) datagramák is terhelik a hálózatot, de talán még nagyobb kiesést jelent a nyugtára való várakozás. A window mezőben azt közli a vevő, hogy hány bájtot küldhet az adó anélkül, hogy visszaigazolást kellene kapnia a vevőtől (általános elnevezéssel ez *burst* eljárás).

Más megközelítésben a WINDOW, a TCP által használt adó és vevő ablak méretét határozza

6 Alapértelmezés az 536 bájtos szegmens adatmező, azaz a szegmens mérete 556 bájt.

7 A 0 kezdőérték a valóságban nem igaz (lásd three-way handshake), mert a kezdőérték egy véletlen szám. (Most azonban fogadjuk el, mert általában az elv helyes.)

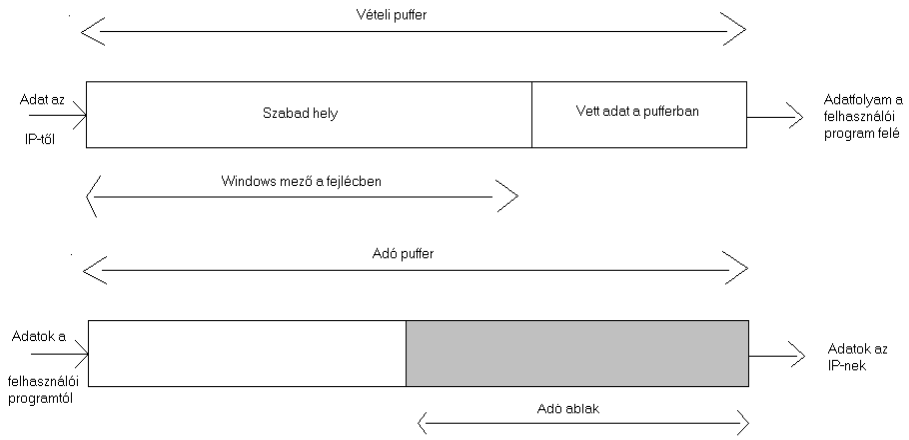
8 Nem kell megijedni, mert általában nem bájtonként küldi a nyugtát. A telnet alatt viszont igen.

9 A hibátlanul megérkezett szegmens sorszáma alapján számítja a nyugtában küldött sorszámot.

meg. Az adási ablakból következik az adó puffer mérete. Itt az olyan sorszámú bájtokat tartják, amelyek már adásra kerültek, de a hozzájuk tartozó nyugta még nem érkezett meg. Az adási ablakban található legkisebb sorszámú bájthoz tartozó nyugta beérkezésekor az ablak előrébb csúszik a következő nyugtázatlan bájtig. A vételi ablak mutatja, hogy milyen sorszámú bájtokat fogadhat el a vevő. A vételi ablakban "látható" sorszámok a beérkező bájtoknak megfelelően növekszenek, azaz az ablak egyre fentebb "csúszik".

Itt térjünk vissza az összeköttetéses kapcsolat fontos részletére, a kézfogási folyamatra. Amikor az adó elküld egy szegmenst, akkor elindít egy időzítőt. Ha az időzítés letelte előtt megérkezik a nyugta, akkor mehet a következő szegmens. Ha nem érkezik időben visszaigazolás, akkor újra elküldi az előző szegmenst. A várakozás miatt a kézfogási folyamat erősen lassítja az adatforgalmat. A WINDOW paraméter (összességében) csökkentheti a várakozásra fordított időt.

Egyébként egy gyors adó képes olyan ütemben tölteni a vevő pufferét, hogy a felhasználói program nem képes kellő sebességgel feldolgozni a beérkező adatokat. A vevő a **WINDOW** mező 0 tartalmával képes az ilyen gyors adókat „megállítani", így ez a TCP **adatfolyam vezérlő** mechanizmusa (flow control).



Összefoglalás: adáskor megengedett, hogy az adó bizonyos számú, átmenetileg nyugtázatlan szegmenst továbbítson. Az ezekben továbbított bájtok száma az ablakméret vagy más néven csúszó keret (sliding window) bájtban mért méretétől függ. Ha egy szegmens nem érkezik meg vevőbe, akkor a vevő a nyugtázó keretben a hiányzó szegmens újraküldését kéri az adótól. Ez a funkció a TCP forgalomszabályzó (Flow Control) mechanizmusához tartozik.

HLEN: (Header Length): azt mutatja, hogy a TCP fejrész hány 32 bites szóból áll és az OPTION mező miatt van rá szükség. Másképpen megfogalmazva az adatok kezdetét jelzi a TCP szegmensben (Data Offset). Értéke 5, ill. opció esetén 6. (Bizonyos esetekben a TCP fejléc méretét csökkenteni lehet 5 bájtra az ún. Van Jacobsen módszerrel, lásd RFC 1144.)

OPTION: ebben a mezőben olyan adatokat helyezhetünk el, amelyek a szabványos fejlécben nincsenek benne. Pl.: a TCP-TCP kapcsolatban itt adható meg az egyes entitások által elfogadható maximális szegmensméret.

RESERVED: 6 db 0 értékű bit.

CODE BIT: 6 bit:

CWR	ECN	URG	ACK	PSH	RST	SYN	FIN

Általában az aktív (magas szintű) bitek azt jelentik, hogy figyelni kell a hozzájuk tartozó adatmezőt. (CWR és ECN bitekkel nem foglalkozunk ebben a jegyzetben.)

Ha az **URG** bit magas, akkor az **Urgent Pointer** (sürgősségi mutató) által jelölt bájttal bezárólag sürgősségi feldolgozást kér a vevőtől. A mutató a sorszámhoz (**SEQNo)** képest mutatja, hogy hol végződik a szegmensben az azonnal feldolgozandó adat. A sürgősséggel feldolgozandó adatoknak a kezdetét az adott alkalmazásnak kell megtalálnia. Ez a modell

hasonlítható egy megszakítási rendszerhez, mert mind az adó és vevő TCP várakozás nélkül kezeli az ilyen jelzésű szegmenseket.

ACK bit: 1-es értéke jelzi, hogy az **ACKNo** mező érvényes adatot tartalmaz. Nyugtázó szegmensben használják. 0 értékénél az **ACKNo** mező figyelmen kívül hagyható.

PSH (Push): pufferelés nélkül (tehát azonnal) kell elküldeni az adatokat, illetve a vevő azonnal átadja az alkalmazásnak[10]. Egyébként általában a TCP nem "zavarja" minden beérkező szegmenssel az alkalmazást, gyakran egy pufferben gyűjtögeti a szegmensek adattartalmát és amikor „eléggé" megtelt a puffer, akkor adja át az alkalmazásnak az adatokat. Ennek ellenpéldája a TELNET alkalmazás, ahol egy TCP szegmensben egy karaktert (!) továbbítanak. A PSH bit kiváltja az azonnali ACK-t a vevőből. Másik értelmezése szerint a sávon kívüli jelzéseket is ezzel a bittel jelölik.

RST: Reset the connection. Több jelentése lehet:

- Egy összeköttetés helyreállításának kezdetét jelzi.
- Más esetben az összeköttetés létesítésére irányuló kérés visszautasítását is jelentheti.
- Nincs a megadott portszámon szolgáltatás (nincs a socket nyitva)
- A jelentkező már nem fér be a várakozási sorba[11].
- A kapcsolódást kérő IP címről nem engedélyezett a szolgáltatás elérése.

Később majd látjuk, hogy a PSH és más biteknek a számítógépen aktív szolgáltatások felderítésénél is nagy szerepe van. (Pl. a hackerek ún. port szkennerekkel fürkészik, hogy milyen szolgáltatások érhetők el egy adott gépen, hogy később egy kiválasztott szolgáltatás biztonsági réseit kihasználva a gép erőforrásaihoz férjenek.)

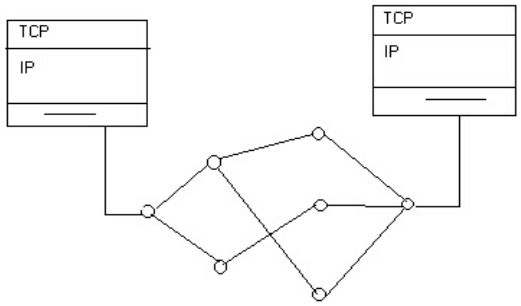
SYN: (Synchronise sequence numbers). Összeköttetés létesítésére szolgál.

FIN: Az adó jelzi, hogy nincs több adat. Bontható az összeköttetés ezen iránya, azaz aki ezt küldte az már nem kíván adni, de venni még átmenetileg szeretne.

Ellenőrző összeg: Szerepe az, hogy a vevő ellenőrizhesse a TCP szegmens hibátlanságát[12].

TCP összefoglalás

A TCP **megbízható, összeköttetés alapú** protokoll, amely duplex összeköttetést biztosít két pont között. (Tehát **nem** támogatja a broadcast és multicast jellegű címeket.) Fogadja az alkalmazási rétegtől a hálózati továbbításra szánt üzeneteket. Ezeket a tetszőleges hosszúságú üzeneteket feldarabolja és TCP fejléccel ellátva átadja a hálózati rétegnek továbbításra, ahol az IP gondoskodik arról, hogy a célgépre valamilyen úton megérkezzenek a szegmensek. A célban az IP átadja a leszállított szegmeseket a TCP-nek, amely azokat sorba állítja (hiszen az IP nem garantálja a sorrendet) és az esetleg hiányzókat pótolva az üzenetet átadja az alkalmazói rétegnek. Az eltérő sebességű adó és vevő között szükség esetén **forgalomszabályozás**t végez. Az adattovábbítás logikai ill. fizikai vázlata:

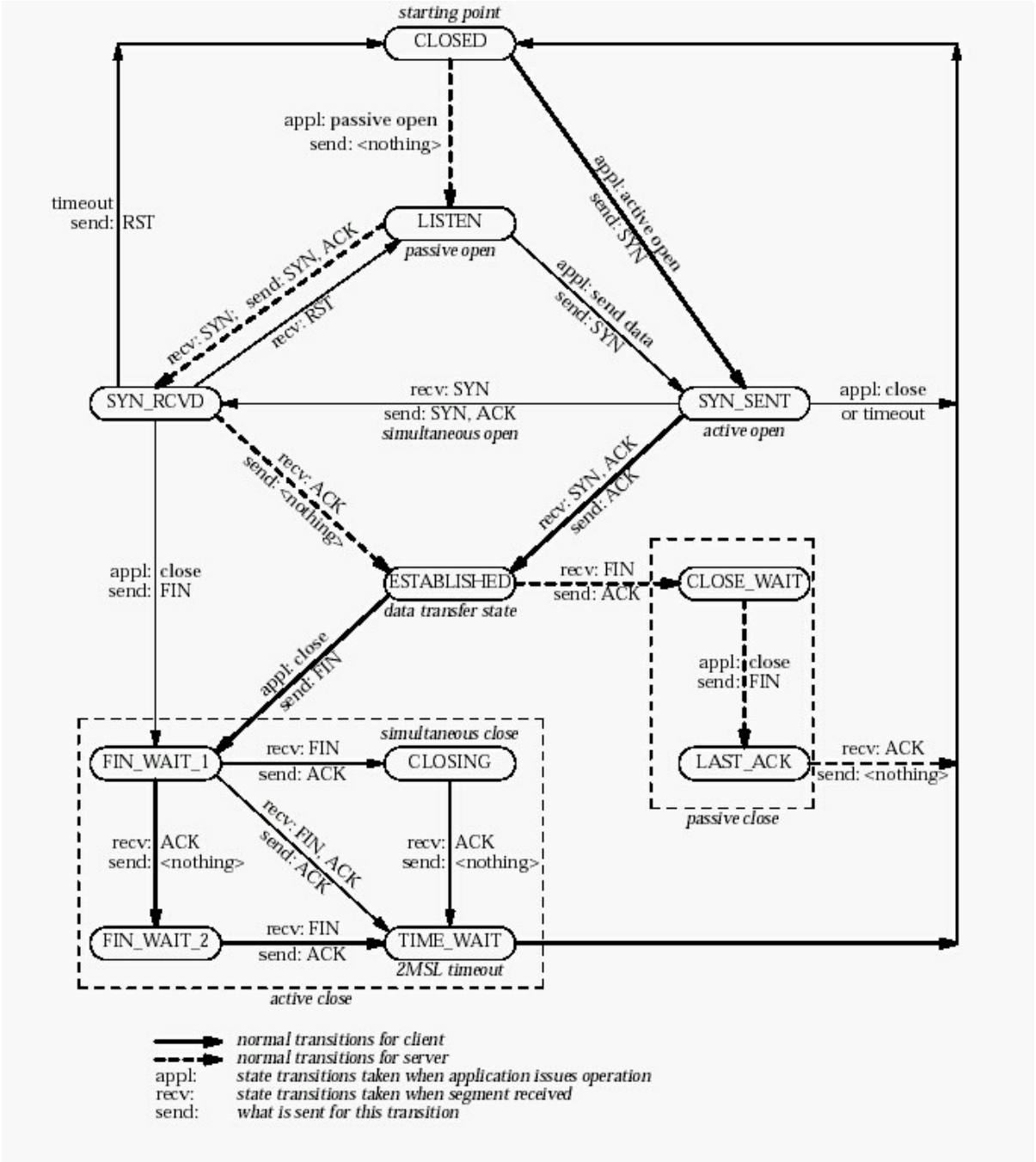


10 Pl. a billentyűzetről bevitt parancsokat ne állítsa be a feldolgozási sorba, hanem azonnal hajtsa végre. A registry-ben beállítható, hogy az adatszegmens PSH bitje aktív legyen-e.

11 Lásd a WinSock **listen** függvényének backlog paraméterét!

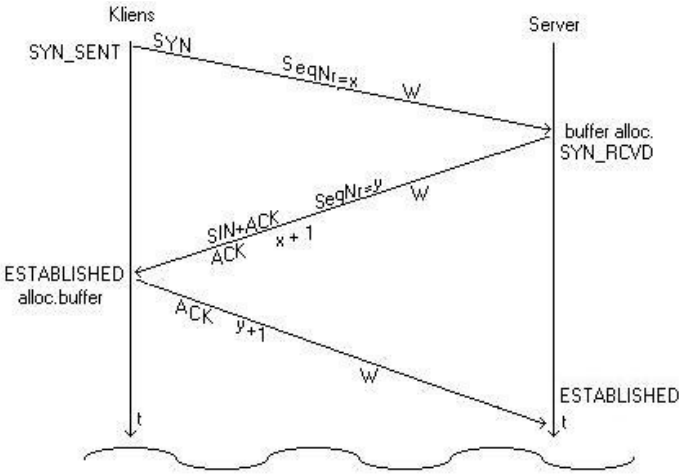
12 Számítása: Stephen A.Thomas: IP kapcsolás és útválasztás (82.old.). Kiskapu 2002

TCP állapotdiagram[13]



Az állapotdiagram könnyebben megérthető, ha részekre osztjuk. Vizsgáljuk először a kapcsolat felvételt (**three-way handshake**):

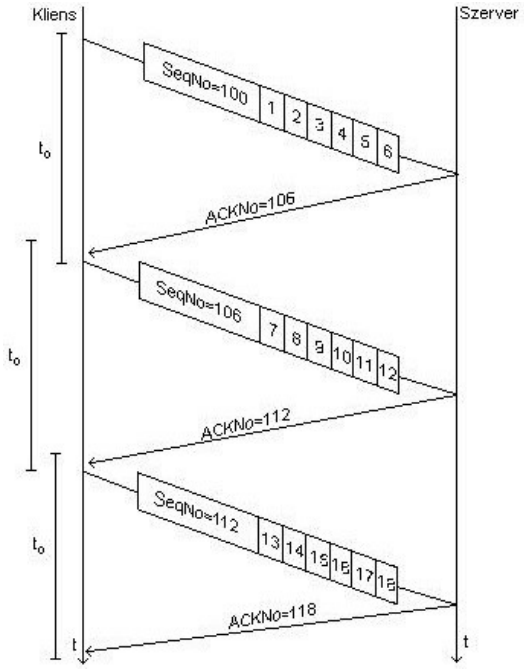
13 http://www.soi.wide.ad.jp/class/20010012/slides/08/19.html



A kapcsolat kialakítását a TCP protokoll szerint a kliens kezdeményezi. A kapcsolat létrejöttét az ún. „three way handshake" eljárás biztosítja. A handshake első fázisában a kliens küld egy olyan szegmenst a szervernek, amelyben a SYN bit magas és ezzel együtt megadja, hogy milyen SEQNo értéktől sorszámozza az általa elküldött adatokat. A SEQNo kezdőértékét véletlenszerűen választja meg biztonsági okokból, legyen most X.

A szerver nyugtázza a kapcsolatfelvételi kérést egy olyan szegmenssel, amelyben az ACK és SYN bitek vannak beállítva. Ezek mellett az ACKNo értékét X+1–re állítva jelzi, hogy a vétel sikeres volt és a kliens legközelebb az X+1 SEQNo értéket használhatja. (Röviden: a kliens SEQNo=X azonosítójú üzenete rendben megérkezett.) A szerver ugyanebben a szegmensben azt is jelzi, hogy az ő SEQNo értéke Y-ról indul. (Y értéke szintén véletlen szám.)

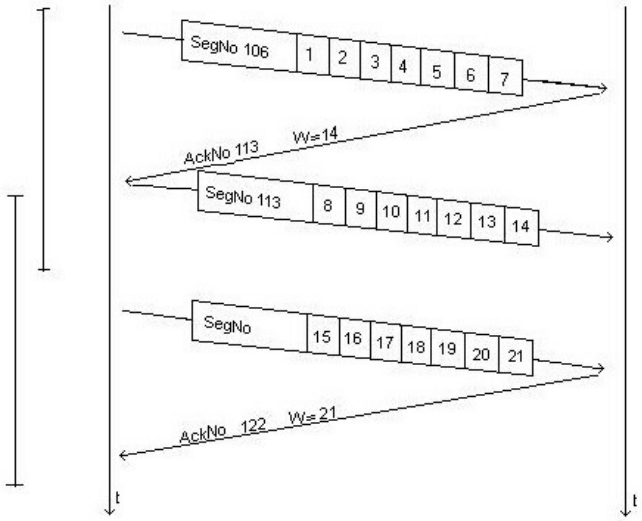
A kliens nyugtázza a szerver előbbi szegmensét egy olyan szegmenssel, ahol az ACK=1 és ACKNo=Y+1. Ezek után a kliens és szerver ESTABLISHED állapotba[14] kerül és ekkor megindulhat a tényleges adatforgalmazás a csatornán:



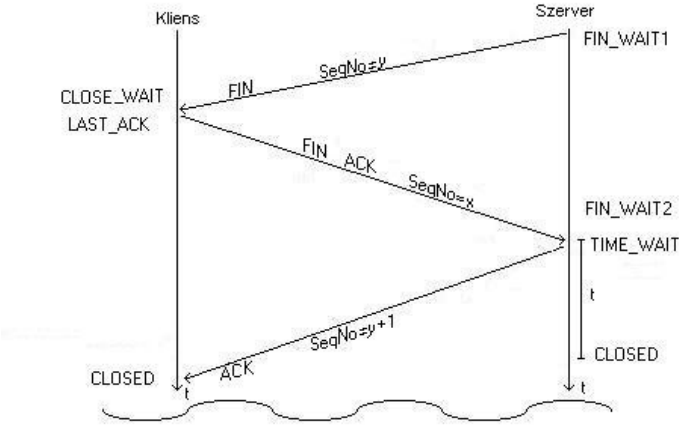
14 A fenti ütemdiagram tehát szoros kapcsolatban van a TCP állapotdiagramjával, ezért még a résztvevők állapotait is feltüntettük: SYN_ SENT (SYN küldés), SYN RCVD (SYN vétel), ESTABLISHED (stabilizált).

Tegyük fel, hogy a three-way handshake kezdetekor a kliens a SEQNo=99 kezdeti értéket használta, amit a szerver nyugtázott. Ezért kaphatta az első adatbájt a SEQNo=100 sorszámot. Amennyiben a nyugtázás a (t_0 maximális) várakozási időn belül megérkezik, akkor a kliens új adatot küld a szervernek. Látszik azonban az is, hogy hibamentes esetekben feleslegesen lassítja az átvitelt az, hogy új küldés előtt feltétlenül várni kell az előzőleg küldött szegmens nyugtázására.

Növeli a teljesítményt, ha a csúszóablakos kézfogásra térnek át. Az előző rajz második szegmensének vétele után a szerver megnöveli a vételi ablakának méretét két szegmens méretűre (W=14). Amennyiben az is sikeres, akkor három szegmens méretűre növel (W=21):



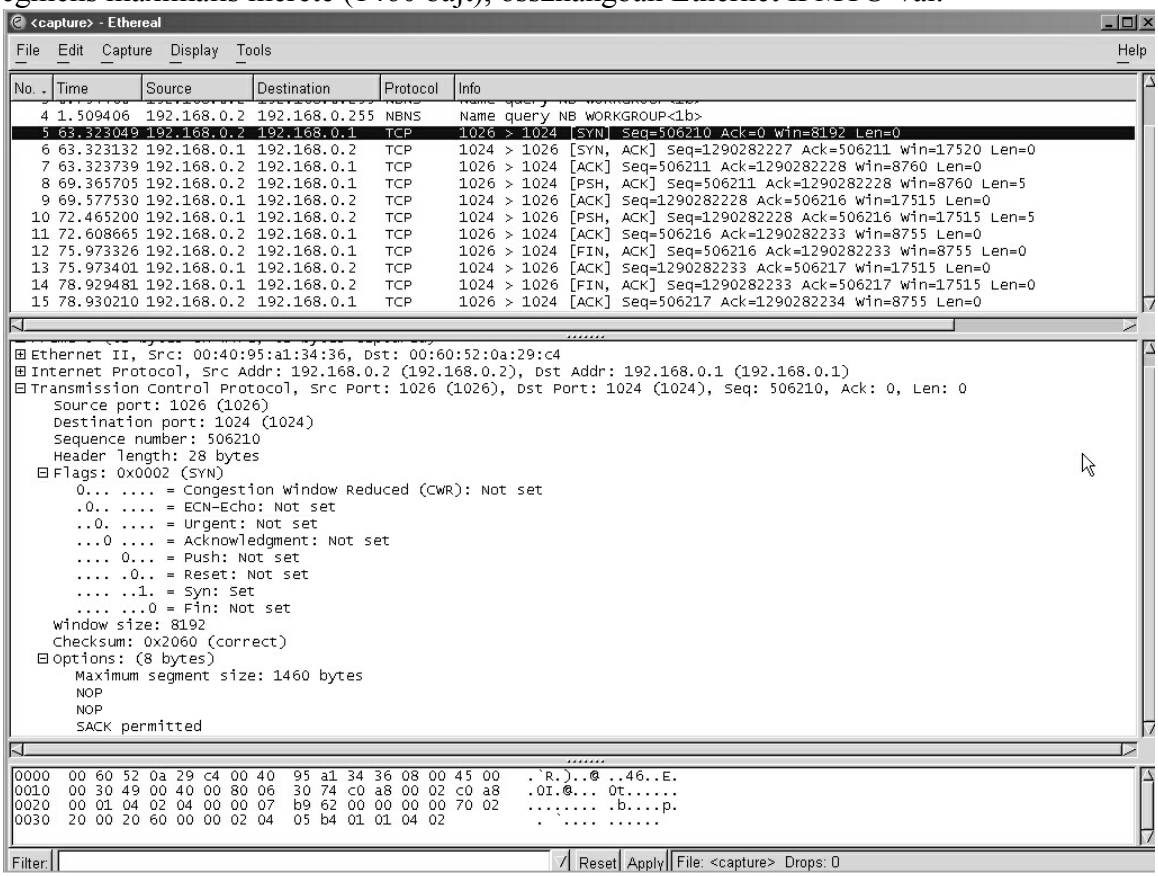
A rajzon tehát adaptív csúszóablak beállítás látható. A módszer szerint, amíg a várakozási idő a max. várakozási időn belül van, növelik az ablak méretét. (Ezzel hibamentes átviteli úton nagyobb állományok átvitele relatíve gyorsabb lesz.) Az ACK, hiba esetén az utolsó hibátlanul vett bájt után következő bájt újraküldést kéri (cummulative ACK)



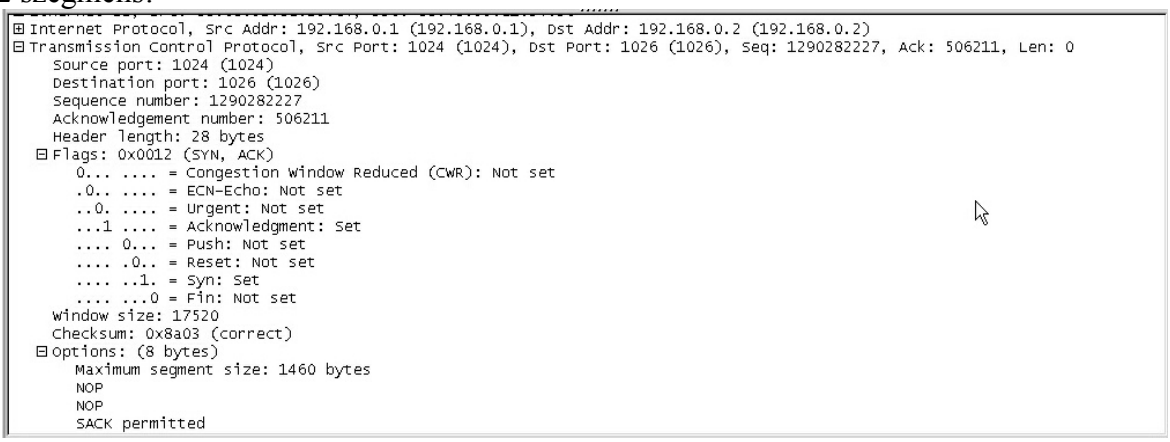
A rajzon példaként a szerver kezdeményezi a kapcsolat bontását (a kezdeményező oldalt aktív oldalnak hívjuk). Adatok nélküli TCP szegmensek mozognak, mert csak vezérlőbitek fontosak. A kezdeményező oldal FIN szegmenst küld olyan sorszámmal, amely az előző forgalmazás után következik. A FIN-t érzékelő oldal a FIN és ACK bittel küldi egy nyugtázó szegmenst, amelyben a sorszám is lényeges. A kezdeményező ezt egy ACK–val nyugtázza. Bizonyos TCP implementációkban a passzív oldalról érkező ACK+FIN jelzés két szegmensben kerül elküldésre.

Egy TCP kapcsolat analízise

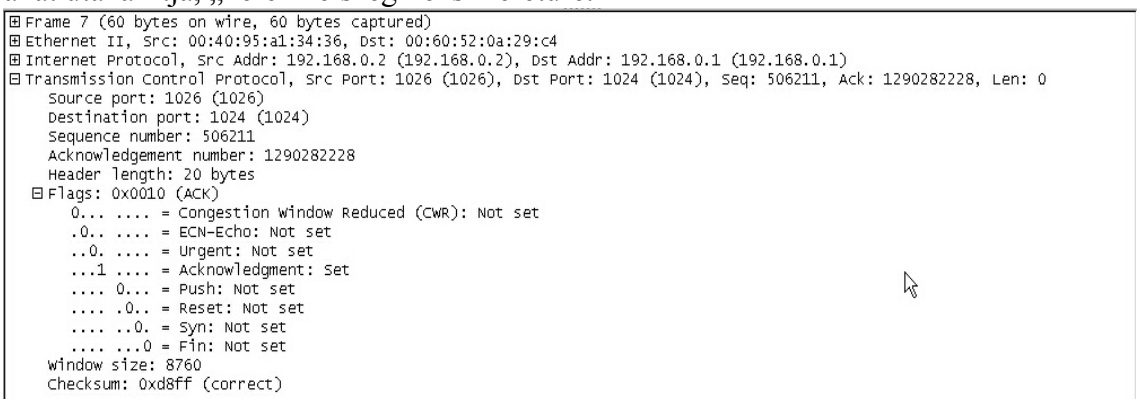
A vizsgált TCP kapcsolatban összesen 11 szegmens került a hálózatra. A szerver (egy XP) az 1024-es portot használta, a kliens (egy W98) pedig a 1026-os portot. Ezeket egyenként megvizsgáljuk. A kapcsolatot a kliens kezdeményezi a SYN bittel, amely azt jelzi, hogy a kezdő sorszám (seq=506210) küldése történik. A ACK bittel érvényesített (ack=0) értéke azt jelzi, hogy a szerver még nem tud mit nyugtázni. A kliens ablaka 8192 bájt és az általa kezelt szegmens maximális mérete (1460 bájt), összhangban Ethernet II MTU-val.



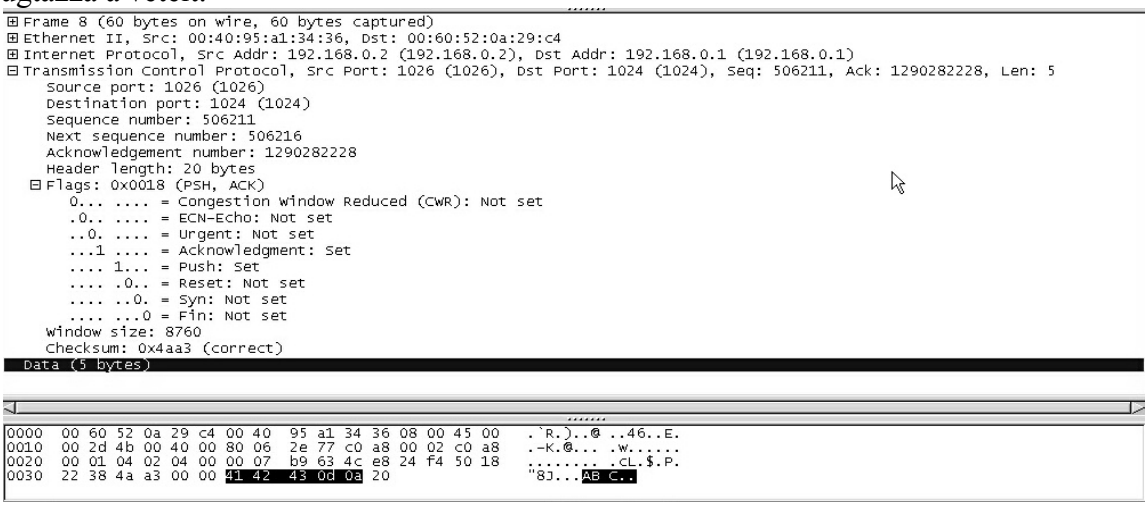
A szerver ack=506211-val jelzi, hogy hibátlanul megérkezett a seq=506210 jelű szegmens és a klienstől a seq=506211-et várja legközelebb. A SYN bittel azt jelzi, hogy a kezdő sorszámát ő is elküldi (seq=1290282227). A szerver által kezelt szegmens maximális mérete (1460 bájt), összhangban Ethernet II MTU-val, viszont TCP szerver ablaka 17520 bájt, amely praktikusan 12 szegmens!



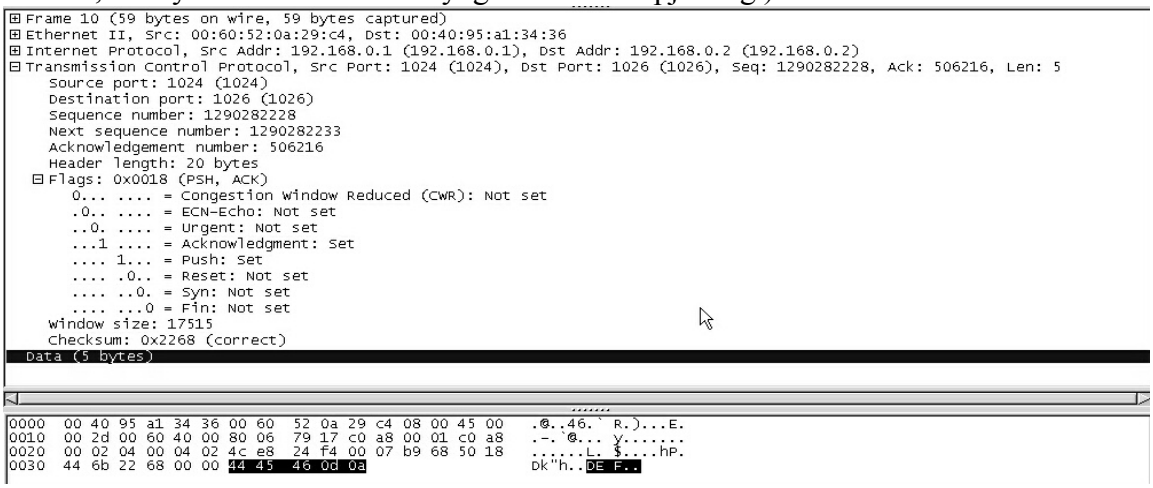
A kliens befejezi a kézfogást azzal, hogy nyugtázza a szerver kezdeti sorszámát és a saját ablakát utánállítja, „kerek” 6 szegmens méretűre.



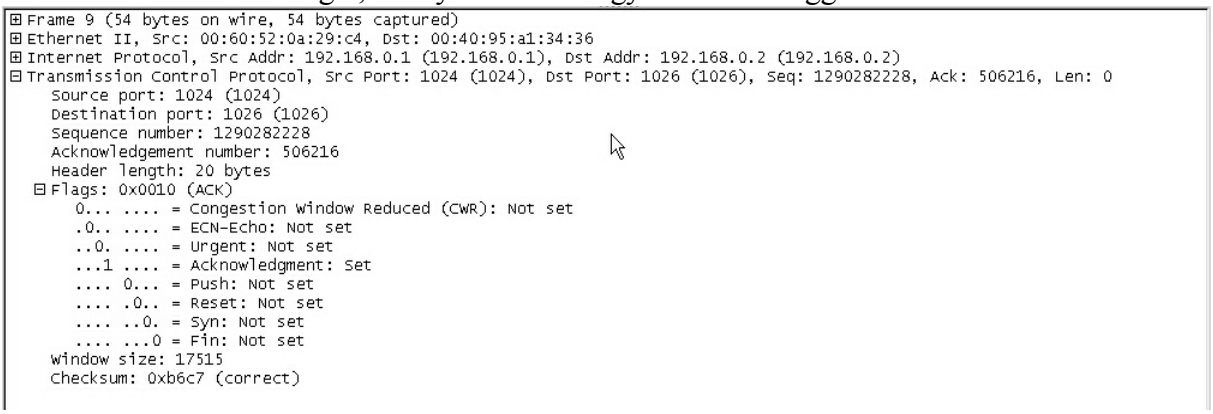
A TCP kliens fölött futó felhasználói „protokoll” 5 bájtos adatként egy sztringet küld a szervernek: ’ABC’<CR><LF>. A PSH jelzi, hogy a vevő ne várja meg az ablakának megteltét, adja át az adatot az alkalmazásnak (amelyet 1024-es port azonosít) és azonnal nyugtázza a vételt.



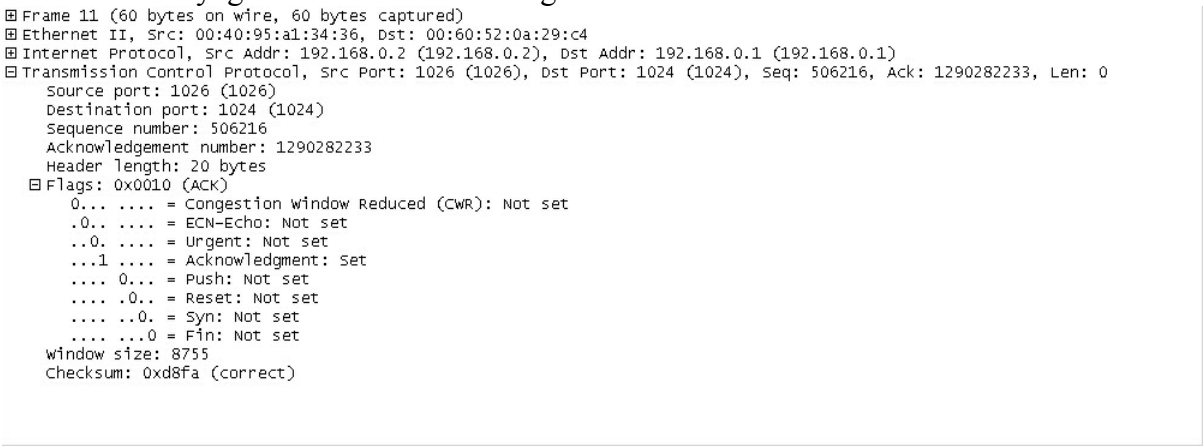
A TCP szerver nyugtázza az adatokat. Az adatok látszólag a vételi bufferben maradtak (ez abból látszik, hogy window 5 bájttal csökkent), mert a címzett alkalmazás még nem ürítette ki. (A TCP a kernel szinten fut, ezért a nyugtázás magasabb prioritású, mint a szerver *recv* metódusa, amely a vezérlést csak a nyugtázás után kapja meg.)



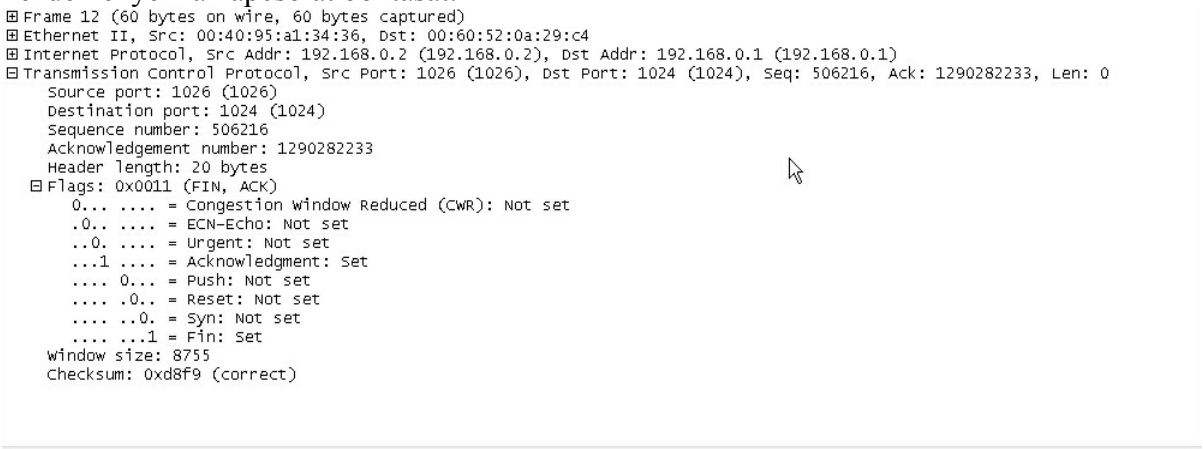
A TCP szerveren futó felhasználói „protokoll” időközben beolvasta a vételi pufferből a ’ABC’<CR><LF> sztringet, amelyre válaszol egy másik sztringgel:’DEF’<CR><LF>



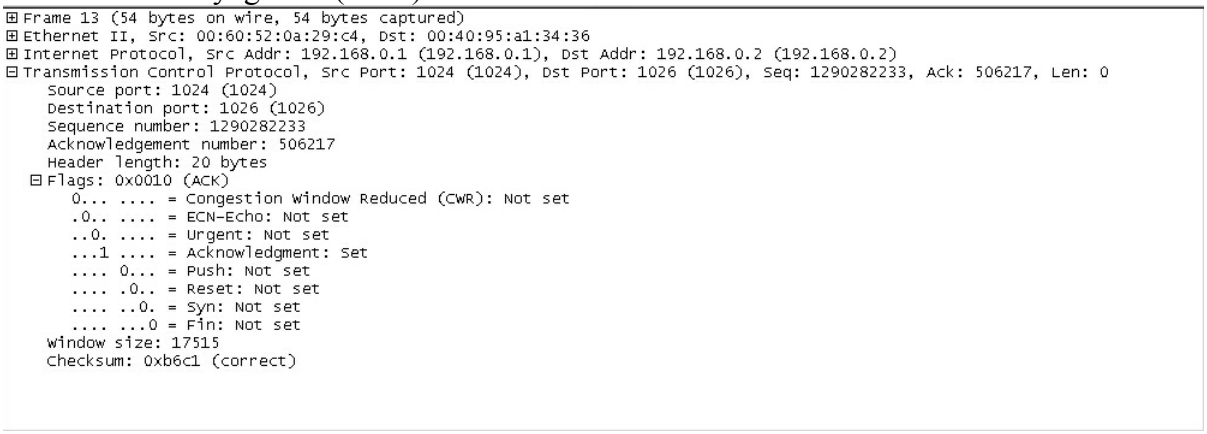
A TCP kliens nyugtáz és vételi buffere még nincs ürítve.



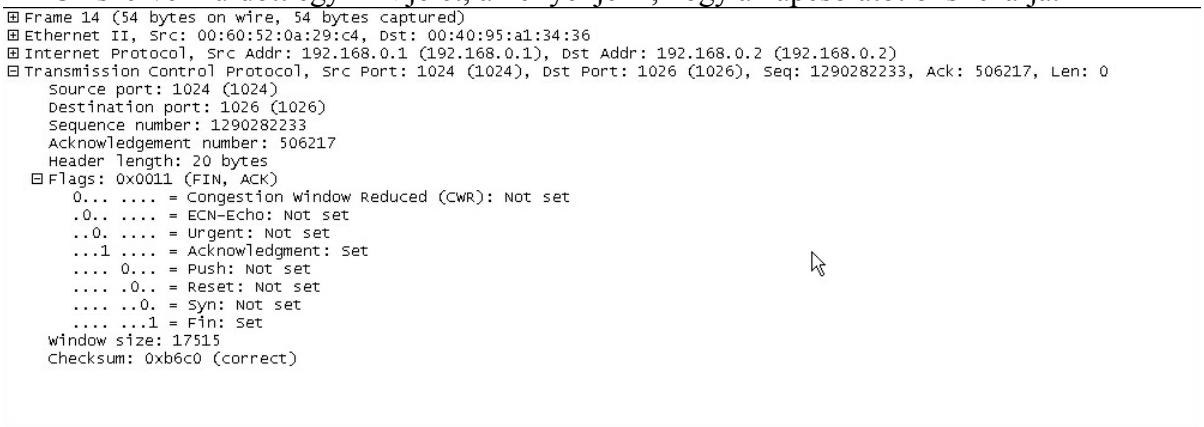
A TCP kliens felhasználói programja meghívja a CloseSocket függvényt, amellyel kezdeményezi a kapcsolat bontását.



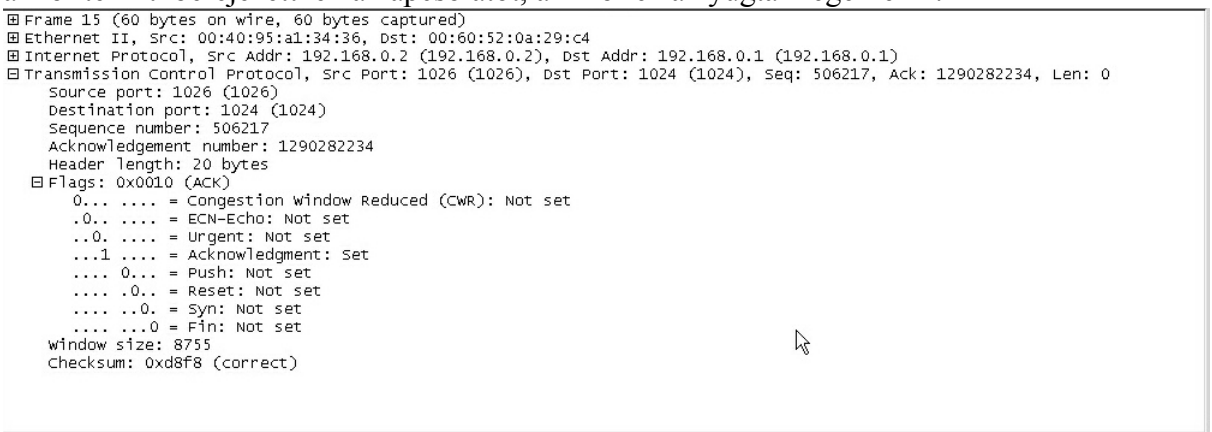
A TCP szerver nyugtázza (ACK) a bontási kérelmet.



A TCP szerver küldött egy FIN jelet, amellyel jelzi, hogy a kapcsolatot ő is lezárja.



A kliens még nyugtázza a FIN-t és ezzel számára befejeződött a kapcsolat. A szerver oldal akkor tekinti befejezettnek a kapcsolatot, amikor ez a nyugta megérkezik.



A TCP szerver-kliens kapcsolatról egy tesztprogram, az állapotdiagram és egy protokoll analizátor segítségével további részletek ismerhetők meg. Mi mélyebbre már nem hatolunk a témában, mert elsősorban felhasználói vagyunk a szállítási rétegnek és nem fejlesztői. Amikor felhasználóként hálózati alkalmazásokat fejlesztünk, a WinSocket API segítségével érjük el a szállítási réteg szolgáltatásait.

USER DATAGRAM PROTOCOL (UDP)

Láttuk, hogy a TCP sajátos tulajdonságokkal rendelkezik, amelyek bizonyos esetekben hátrányosak is lehetnek. Már magának a kapcsolatnak kiépítése is három IP csomagot igényel, és az ACK szegmensre való várakozás még lassítja az adatátvitelt, bizonyos alkalmazások pedig lehet, hogy a multicast címek hiánya miatt nem tudják használni. Az UDP egy **összeköttetés mentes** kapcsolatot biztosít a gépek[15] között. A TCP-vel kialakított datagram folyam helyett egyetlen (korlátozott méretű) datagram továbbítására lett kifejlesztve[16]. A datagramot fogadó gépnek nem kötelező válasz datagramot küldenie. A küldő alkalmazás dönti el azt, hogy ha bizonyos időn belül nem érkezik válasz, akkor újraküldi-e a datagramot vagy pedig valami mást csinál. Sok alkalmazási protokoll működik olyan módon, hogy egy kérésre egy választ küld, amely egyben az ACK funkciót is jelenti. Számukra tökéletes az UDP. Az UDP fejléc láthatóan egyszerűbb, mint a TCP fejléc:

Adó port $_{(16\ bit)}$	Cél port $_{(16\ bit)}$
Hossz $_{(16\ bit)}$	Ellenőrző összeg $_{(16\ bit)}$
Az alkalmazási réteg adatai $_{(n*\ 8\ bit)}$	

Adó és cél port: A TCP-hez hasonlóan az UDP-t használó alkalmazások azonosítását biztosítja. A portszámokat felsoroló állományban egy adott portszám mellett megtaláljuk mind a TCP-re mind az UDP-re épülő alkalmazói protokollt. (Pl.: DNS=53, DHCP 67/68)
Hossz: A fejléc és az adatok együttes hossza. Másképpen az UDP szegmens hossza.
Ellenőrző összeg: Kiszámítása hasonló, mint a TCP-nél alkalmazott ellenőrző összeg esetén. Használatával a vételi helyen ellenőrizhető az UDP szegmens hibátlansága.
Adatok: Az alkalmazói réteg adatai.

Összefoglalás: UDP használatakor a szállítási rétegtől kevés szolgáltatást kapunk. Célunk inkább az IP szolgáltatásainak közvetlenebb (gyorsabb) elérése lehet.

Kérdések
Ismertesse a következők jelentését: port, TPDU, en- és decapsulation, szegmens!
Ismertesse a TCP fejléc mezőinek szerepét!
A portszámok milyen kategóriákba sorolhatóak és mi jellemzi ezek használatát?
Soroljon fel hálózati alkalmazásokat, amelyek a kliens-szerver modellt követik!
A kliens-szerver modell alkalmazásakor milyen megfontolás alapján választanak portszámot és melyik fél kezdeményezi a kapcsolatot?
Ismertesse a „Három fázisú kézfogás” folyamatát!
Mi az URL szerepe és felépítése?
Magyarázza meg a „pozitív nyugtázás újraküldéssel” fogalmát!
Hogyan segíti a TCP az eltérő feldolgozási sebességű számítógépek kapcsolatát?
Mit takar az ablakméret és a „csúszó ablak” fogalma? Mi azonosítja a szegmenseket?
Hogyan „előzhet” a vevő oldal bufferében egy szegmens?
Milyen célra előnyös a TCP és milyen célra az UDP? Milyen funkció közös bennük?
Mit jelent az összeköttetéses és összeköttetés mentes kapcsolat?
Hogyan jellemezné a szegmensek útvonalát és beérkezési sorrendjét?
Hogyan vizsgálná meg a gépén az egyes alkalmazások által használt portokat?

15 Pontosabban az UDP entitások között.
16 Így nem kell az időt arra használnia, hogy összeköttetést alakítson ki a gépek között. Tehát a háromutas kézfogás elmarad.