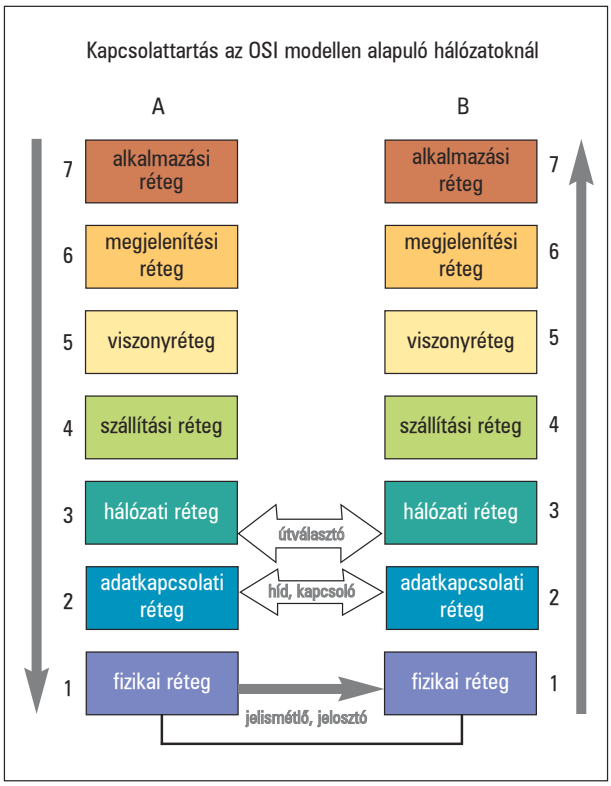
Hálózatok (2. rész)

Sorozatunk e részében szó lesz az entitásokról, a csatolófelületekről, a protokollokról, a hivatkozási modellekről és sok minden másról.

Mint azt az előző részből megtudtuk, a rétegeknek az a feladatuk, hogy valamiféle feladatot végezzenek el a közvetlenül felettük álló réteg számára. Ezt úgyis mondhatjuk, hogy az n. réteg szolgáltatást nyújt az n+1. réteg számára.
Felmerülhet a kérdés, hogy miként nyújthat egy réteg szolgáltatást? Valójában nem maga a réteg az, amely szolgáltatást nyújt, hiszen a réteg nem kézzelfogható, inkább csak amolyan elméleti dolog. Minden réteghez tartoznak azonban olyan elemek, amelyek „fizikai” értelemben véve is munkát végeznek. Ez lehet például egy folyamat, de lehet egy eszközön elhelyezett nyomtatott áramkör is – ezek az elemek az adott réteg entitásai (entitites).
Átfogalmazva tehát azt is mondhatjuk, hogy egy adott réteg entitásai szolgáltatásokat végeznek a közvetlenül felettük lévő réteg számára. Ha két entitás ugyanabban a rétegben, ám különböző gépen helyezkedik el, akkor ők úgynevezett társentitások (peer entitied).
Az odáig rendben van, hogy a rétegek entitásai képesek bizonyos feladatokat elvégezni, ám a felső rétegeknek valamiképpen el kell tudniuk érni őket, ezért szolgáltatáselérési pontokra (Service Access Points, SAP) van szükség, amelyeken keresztül az alsó rétegek szolgáltatásai elérhetőek. Fontos az is, hogy ezek a pontok egyértelműen megnevezhetőek legyenek, ezért mindegyiknek egy egyedi címmel kell rendelkeznie.
A SAP csak első hallásra tűnhet titokzatosnak, valójában egyszerű dologról van szó. Vegyünk például egy szinte minden háztartásban megtalálható hálózatot: a vonalas telefonhálózatot. Ennél vajon mi lehet a szolgáltatási pont? Természetesen nem más, mint az a falba szerelt háromlábú csatlakozó, amihez a telefonkészülékünket kapcsoljuk. Ezek a SAP-ok egyedi címmel is rendelkeznek, amiket telefonszámoknak nevezünk.

Csomagok

Azt már tudjuk, hogy két szomszédos réteget az úgynevezett csatolófelület (interface) köt össze, de az még rejtve maradt számunkra, hogy miként is működik a rétegek közötti adatcsere. Ahhoz, hogy egy csatolófelületen adatokat adjunk át, szükség van bizonyos előzetes megállapodásokra. A rétegek közötti csatolófelületek általában a következőképpen működnek: a felső réteg entitása az alsó réteg szolgáltatáselérési pontján keresztül az alsó réteg entitásának egy úgynevezett csatolófelület-adategységet (Interface Data Unit, röviden IDU) ad át. Az IDU két részből áll: a vezérlőadatból és egy szolgáltatási adategységből (Service Data Unit, azaz SDU). Az SDU tulajdonképpen maga az adat, amit a hálózaton – egész pontosan a társentitásnak – el szeretnénk küldeni. A vezérlőadat azonban nem része ennek az üzenetnek, hanem magáról az átküldendő adatról árul el bizonyos dolgokat, amelyekre az alsó rétegnek szüksége lehet (vagy legalábbis megkönnyíti a munkáját). Vezérlőadat lehet például az SDU mérete.
Az SDU továbbításának módja az alsó rétegre van bízva. Ha túl nagy méretű, akkor az alsó réteg akár fel is darabolhatja, majd fejléccel ellátva darabonként küldheti tovább. A fejléc vezérlőadatokból áll és az SDU elejére illesztik. Az ilyen fejléccel ellátott darabkákat protokoll-



1. ábra Kapcsolattartás az OSI modellen alapuló hálózatoknál

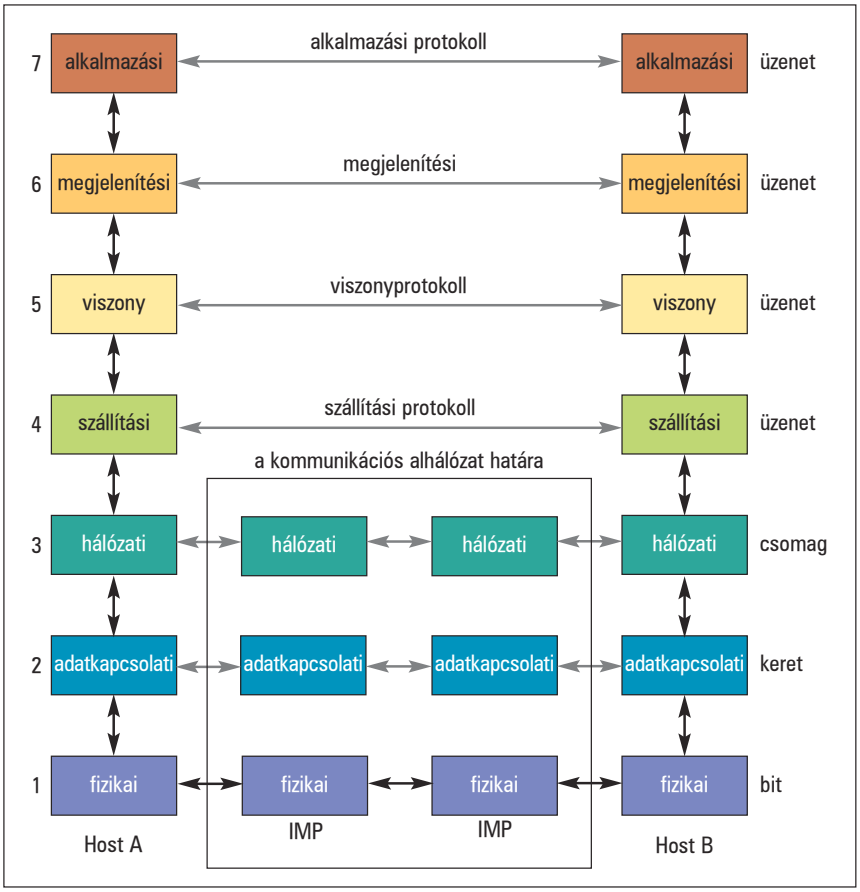
adategységeknek (Protocol Data Units, röviden PDU) nevezzük. Fontos, hogy a PDU-k fejléce csak a társentitásokat érdekli, a felsőbb rétegekhez már nem jutnak el. Az interneten áramló csomagok is egyfajta PDU-k.

A rétegek szolgáltatásai

Az alsó réteg a felső réteg szempontjából mindig arról gondoskodik, hogy az általa küldött üzenetet a társentitásnak átjuttassa. Erre azonban sokféle út kínálkozik, az egész attól függ, hogy milyen jellegű a kapcsolat. Most röviden átbeszéljük ezeket a szolgáltatástípusokat – érdemes nagyon figyelni, mivel a későbbiekben sokat fogunk rájuk hivatkozni.

Mind közül az első a kapcsolatközpontú szolgáltatás (connection oriented service); legfontosabb tulajdonsága, hogy az átvitel megkezdése előtt fel kell építeni a kapcsolatot. Miután ez létrejött, az adatok sorban továbbíthatók (amelyek a továbbítási sorrendben is érkeznek meg), majd ha minden kész, akkor ugyanazzal a lendülettel le is kell bontanunk a kapcsolatot. Ez olyasmi, mint a telefonálás: először fel kell hívnunk az illetőt, és miután mindent megbeszéltünk vele, le kell raknunk a kagylót.

Ennek az elvnek gyökeres ellentéte a kapcsolat nélküli szolgáltatás (connectionless service). Ekkor nem kell a kapcsolat felállításával bajlódnunk, az adatcsomagokat egyszerűen csak a megfelelő címre kell küldenünk. Az ilyen jellegű szolgáltatásoknál általában minden üzenet más útvonalon halad, és sohasem lehet teljes bizonyossággal megbecsülni az átviteli időt. Sőt olyan eset is előfordulhat, hogy a másodiknak küldött csomag előbb ér oda, mint az első. Jellemzően ilyen kapcsolat az elektronikus levél vagy a hagyományos levelezés.



2. ábra A rétegek ugyanazok

A szolgáltatásokat azonban más jellemzők alapján is csoportosíthatjuk, például a minőségük alapján. Egyes szolgáltatások teljesen megbízhatóak, azaz minden elküldött adat sértetlenül és biztosan megérkezik. Akadnak azonban olyan szolgáltatások is, amelyeknek nem éppen a megbízhatóság az erősségük. Például könnyen előfordulhat, hogy bizonyos bájtok elvesznek, illetve sérülten érkeznek.

Ne becsüljük le azonban az utóbbi típusba tartozó szolgálatokat sem! Egy megbízható kapcsolathoz ugyanis elengedhetetlen, hogy a vevő fél valamiféleképpen ellenőrizni tudja a kapott üzenetek sértetlenségét; ezenkívül gondoskodni kell a hibajavításról is. Egyes kódolási eljárások segítségével bizonyos hibákat a vevő önállóan is ki tud javítani, de az is előfordulhat, hogy meg kell kérnie a küldőt: ismételje meg az üzenetet. (A hibajavító kódokról egy későbbi részben bővebben is szót ejtünk.) Ez azonban még mindig kevés. Ha teljesen bizonyosak akarunk lenni, akkor a vevő félnek minden kézhez kapott üzenet után egy nyugtát kell visszaküldenie. Miután a nyugta visszaérkezett, akkor a küldő biztos lehet benne, hogy az üzenet célba ért, és küldeni lehet a következő adatblokkot.

Könnyű belátni, hogy az ilyen, nyugtázáson alapuló kapcsolat jelentősen növeli az adatforgalmat. Előfordul azonban az is, hogy az ebből adódó késleltetés nagyobb gondok forrása lehet, mintha egy-egy bájt hibásan érkezne meg (vagy esetleg meg se érkezne). A legjobb példa erre a hálózaton keresztüli hang- és videólejátszás. Ha minden elküldött csomag után várnánk a nyugtát, a hang, illetve a kép szaggatott lenne, ami az élvezhetőség rovására menne. Ha azonban néhány képpont hibásan érkezik, az nem annyira zavaró (már ha egyáltalán feltűnik), mintha az egész filmet szaggatottan kellene végignéznünk.

Egy állomány átküldésénél azonban más a helyzet: ott egy bitet sem szabad tévednünk, a legkisebb átviteli hiba is végzetes lehet. Ebben az esetben nem tarthatjuk fényűzésnek a nyugtázást, mivel ebben az esetben a sebesség kérdése csak másodlagos.

Az ilyen megbízható kapcsolatalapú szolgálatásokat két altípusra bonthatjuk tovább: az egyik ezek közül az üzenetsorozat. Itt létezik egy előre megbeszélt blokkméret – az üzenetet a küldő ilyen méretű csomagokra darabolja, majd egymás után elküldi őket. Ha ez a méret például 1 KB,

akkor egy 2 KB-os üzenet két darabban fog megérkezni.
A másik fajta a bájtfolyam – ez olyasmi, mintha kinyitnánk egy csapot. Két elküldött adatmennyiség között nem létezik semmiféle határ. Ez a megoldás a küldő szempontjából biztosan kényelmesebb, a vevő azonban képtelen lesz megállapítani, hogy a kézhez kapott 2 KB az két 1 KB-os vagy egy 1 KB-os üzenet. Bájtfolyamot jellemzően a terminálok használnak.
A kapcsolat nélküli szolgáltatások között is van megbízható és kevésbé megbízható. Az utóbbit datagram szolgáltatásnak nevezzük, megbízhatóbb formáját pedig nyugtázott datagram szolgáltatásnak (acknowledged datagram service).
Ezt akkor érdemes használni, amikor csak valami apró méretű dolgot szeretnénk átküldeni. Így ugyanis a kapcsolatot nem kell felépítenünk és lebontanunk, ami már önmagában is tovább tartana, mint maga az üzenet elküldése.
Létezik még egy szolgáltatás, amelyről illik pár szót ejteni. Ez nem más, mint a kérés–válasz, amelyet az ügyfél–kiszolgáló modellben használnak. Ekkor a küldő egy kérdést küld el, amelyre a visszajelzés maga lesz a válasz, például ilyen, ha egy adatbázis-kiszolgálótól lekérjük egy meghatározott rekord tartalmát.

Alacsonyszintű szolgáltatás

Már majdnem mindent tudunk a hálózatok elvi működéséről, egyedül az alacsonyszintű szolgáltatásoról nem szóltunk. Hogy ez pontosan mit takar, azt nagyon nehéz megfogalmazni (legalábbis csak hihetetlenül bonyolultan lehet), pedig egyszerű dologról van szó.
Az alacsonyszintű szolgáltatások nem mások, mint elemi műveletek, amelyeket a szolgáltatás végre tud hajtani és a felhasználó (vagy a felsőbb réteg) meg tud hívni. E műveletek segítségével hajtathatunk végre bizonyos feladatokat a szolgáltatással, illetve kaphatunk adatokat a társentitás állapotáról.
Nézzünk egy példát! Ha mondjuk az a feladat, hogy építsünk ki egy kapcsolatot, akkor először a `CONNECT.request` műveletet kell meghívnunk. Ennek hatására a szolgáltatás egy kapcsolatteremtési és kérelmet küld a címzett számára. Amint az üzenet megérkezik, a vevő entitása az alatta lévő szolgálatatástól egy `CONNECT.indication` jelzést kap. Ekkor az entitás a `CONNECT.response` primitív segítségével elfogadhatja a kapcsolódási kérelmet, de akár vissza is utasíthatja. Ha mégis belemegy, akkor a küldő entitás a `CONNECT.confirm` jelzés segítségével értesülhet a kapcsolat létrejöttéről.
Egyes primitívek értékekkel is bírhatnak, például a `CONNECT.request`-nek értékként meg kell adni a célállomást.

A protokoll

A protokoll és a szolgáltatás között szoros kapcsolat van, ám sokan úgy gondolják, hogy ez a két fogalom egy és ugyanaz. Mivel ez az elképzelés sajnos téves, fontos rávilágítanunk a protokollok és a szolgáltatások közötti különbségekre.
Először is foglaljuk össze, hogy mi is az a szolgáltatás: ha nagyon pontosak szeretnénk lenni, akkor a szolgáltatást egy halmaznak tekintjük, amelynek az elemei a primitívek – ezek a felsőbb rétegek számára nyújtanak bizonyos szolgáltatásokat. Fontos, hogy a szolgáltatás semmit nem árul el a megvalósításról, azaz hogy miként hajtja végre ezeket a feladatokat; csupán egy (elemi) művelethalmaz, amelynek elemeit, a felsőbb rétegek hívhatják meg.
A protokoll azonban nem a műveletek, hanem a szabályok halmaza. Ezek a szabályok pontosan megmondják, hogy a csomagok milyenek legyenek, hogyan nézzen ki a fejléc stb. A protokoll a szolgáltatások megvalósításának eszköze. Fontos azonban, hogy az entitásoknak módjukban áll a protokollokat megválasztani, feltéve, hogy ezáltal nem módosulnak az alacsonyszintű szolgáltatások.
Láthatjuk tehát, hogy a szolgáltatások és a protokoll fogalma élesen elkülöníthető. Voltak olyan hálózatok, amelyek nem tettek efféle megkülönböztetést. Az ilyen hálózatokon a kapcsolattartás valahogy úgy folyt, hogy a folyamat saját maga állította össze a csomagot egy memóriaterületre, majd meghívta a `Send packet` (vagy valami hasonló nevű) rendszerhívást, amelynek az értéke az adott memóriaterület címe volt. Ez azért volt rossz, mert a felhasználónak ismernie kellett az adott protokollt. Ezért, ha meg akarták változtatni a protokollokat, akkor az összes hálózati alkalmazást módosítani kellett.

Hivatkozási modellek – az OSI modell

A hivatkozási modellek nem mások, mint kézzelfogható példák arra, hogy milyen rétegekől állhatnak egyes hálózatok. Mi kettővel ismerkedünk meg: a mostani részben az OSI-val, a következőben a TCP/IP hivatkozási modellel.
Az OSI modellt az iskolákban nagyon szeretik oktatni (és számonkérni), ám gyakorlati haszna nem igazán van. A TCP/IP-nek azonban annál inkább, mivel az egész internet erre épül.
Fontos az elején megjegyeznünk, hogy a TCP/IP és az OSI modell távolról sem tökéletes. Mindenesetre mindkét modellel megismerkedünk, viszont sorozatunkat az OSI modell szerint fogjuk felépíteni. Ennek okaira inkább később derítenénk fényt, most legyen elég annyi, hogy a TCP/IP-ben a rétegek sokkal összetettebb feladatokat látnak el, így az elméleti működést könnyebb vázolni, ha az OSI szerint haladunk. A továbbiakban tehát az OSI rétegeivel ismerkedünk meg.
Az OSI modell történetéről csak annyit, hogy a Nemzetközi Szabványügyi Szervezet (ISO) ajánlására tett egyféle lehetséges megoldás, amelyet első ízben sikerült nemzetközileg is elfogadtatni.
Az OSI modellnek hét rétege van. Most gyorsan átfutjuk, hogy mely rétegek milyen feladatok elvégzésére hivatottak. A legalsó a fizikai réteg, amelynek a feladata a bitek továbbítása az egyik géptől a másikhoz. Itt inkább műszaki nehézségeket kell leküzdeni, például garantálni, hogy pontosan az érkezik meg, amit átküldtünk.
Az adatkapcsolati réteg legfontosabb feladata a fizikai réteg munkájának az ellenőrzése, azaz hogy a legalsó réteg által fel nem ismert hibákat kiszűrje. Gond abból adódik, hogy a fizikai réteg csak bitfolyamokkal dolgozik, a belső adattartalommal nem foglalkozik, s így az esetlegesen megjelenő vonalzajokból fakadó hibákat nehéz kiszűrni. Az adatkapcsolati réteg ezért az elküldendő adatot úgynevezett adatkeretekre (data frames) darabolja fel. A keretek mérete általában legfeljebb pár kilobájt; a réteg a kereteket sorrendben továbbítja. Mivel a fizikai réteg a kerethatárokkal nem foglalkozik, az adatkapcsolati szintnek valahogy el kell tudnia különíteni az egyes rétegeket egymástól; ezért minden réteget egy különleges bitsorozat zár. Előfordulhat olyan eset is, amikor az adattag véletlenül tartalmazza ezt a bit-

sorozatot. Ilyenkor a vevő fél nem lesz képes helyesen összeállítani a kereteket, ami komoly bonyodalmakhoz vezethet. Emiatt arról is gondoskodni kell, hogy a vevő nehogy kerethatárként fogja fel az adattagban megbúvó keretvége-jelzéseket.

Amikor egy keret megérkezik a vevő félhez, annak egy nyugtázó keretet (acknowledgment frame) kell visszaküldenie. Ha a küldő fél nem kap ilyet, akkor nagy a valószínűsége annak, hogy a küldött adat elveszett, és a kérdéses keretet ismét el kell küldeni. Az élet azonban sohasem egyszerű, főleg ha maga a nyugtázó keret eltűnik. Ekkor a küldő nem értesül a keret megérkezéséről, és ismét elküldi, azaz a vevő kétszer kapja meg ugyanazt a keretet (keretkettőzés). Egy másik komoly gond forrása lehet az is, ha az adó gyorsabban ad, mint ahogy a vevő képes lenne feldolgozni. Ezt az esetet csak úgy lehet elkerülni, ha a réteg tartalmaz valamiféle forgalomkezelő módszert. Ilyen lehet például, hogy az adó megkérdezi a vevőt, hogy az éppen mekkora méretű szabad átmeneti tárral rendelkezik.

A következő szint a hálózati réteg, amelynek első számú feladata a csomagok célba juttatása. Ennek elengedhetetlen eleme az útvonalválasztás, azaz minden csomag számára ki kell jelölni egy utat a forrás- és a célállomás között. Az útvonalválasztásnak két fajtája van. Létezik a statikus, amikor a hálózatba bele vannak „égetve” az útvonalak, de lehet dinamikus is. Ebben az esetben az útvonalválasztás a különböző csatornák terheltségétől függően kerül kiválasztásra.

A csomagok célba juttatásához az útvonalválasztáson kívül még egy bonyolult nehézséget le kell küzdeni: a hálózatok közötti együttműködés hiányát. A hálózatok ugyanis különböző csomagmérettel és protokollokkal dolgozhatnak. Ha a cél tehát egy másik hálózat, akkor gondoskodni kell a csomagok helyes átalakításáról is.

A szállítási réteg egyfajta összeköttetés a felhasználó és a hálózati eszköz között; lényegében ez határozza meg, hogy a felhasználók számára az adott hálózat milyen szolgáltatásokat képes nyújtani. Ez és a felette lévő rétegek mindegyike azonban más jelentős dolgokban is különböznek az alsó háromtól.

Ennek a szintnek a feladata hivatalosan csak annyi, hogy a fentről kapott elküldendő adatokat feldarabolja, elküldje a célállomásnak, ami hibátlanul megkapja őket. Az adatkapcsolati réteggel szemben itt az a különbség, hogy már nem kell különböző műszaki nehézségekkel foglalkoznunk.

A gépszint már rejtve marad számunkra. Ezért a szállítási réteg úgymond végpontok közötti kapcsolatot hoz létre. Ez abban különbözik az előbbiektől, hogy azok mindig csak a hálózatban lévő legközelebbi szomszéddal tartották a kapcsolatot. Itt azonban már nem foglalkozunk azzal a ténnyel, hogy egy-egy csomag több csomóponton keresztül jut el a célállomásig.

Mit jelent ez? A hálózatokban mindig két program tart kapcsolatot egymással: a forrásállomás programja kapcsolatot kezdeményez a célállomás valamely hasonló programjával. Miután a kapcsolat létrejött, a két program olyan módon tart kapcsolatot egymással (csomagok és vezérlőjelek segítségével), mintha a két gép közvetlen kapcsolatban állna.

A hálózati alkalmazások számára rejtve marad az alkatrészes megvalósítás, és teljesen mindegy az is, hogy a csomagnak a célig hány csomóponton keresztül kellett átküzdenie magát. Az alsó három réteg ezzel szemben mindig csak a szomszédos állomásokkal foglalkozik.

Amikor egy felsőbb réteg kapcsolatba akar lépni valakivel a hálózaton, akkor a szállítási réteg úgynevezett hálózati összeköttetést hoz létre. Minden egyes kapcsolatnak külön összeköttetése van. Sőt még azt is lehet, hogy a szállítási réteg a gyorsabb átvitel érdekében további összeköttetéseket hoz létre a célállomással, ezáltal lehetővé teszi, hogy az adatokat az egyes csatornák között megossza. Mivel a számítógépek manapság többfeladatos operációs rendszereket futtatnak, a gépek egyszerre több kapcsolatot is fenntarthatnak. Ehhez azonban az is szükséges, hogy az egyik gépen futó program megmondhassa, hogy a másik gép melyik programjával akar beszélni. Hogy a szállítási réteg miként birkózik meg ezekkel a feladatokkal, arról a későbbiekben kimerítő részletességgel fogunk szólni.

A viszonyréteg tulajdonképpen egyfajta irányító szerepet tölt be: egyrészt irányítania kell a kapcsolatot, mivel bizonyos esetekben nem adhat egyszerre mind a két fél, s a viszonyrétegnek kell meghatároznia, hogy mikor ki adhat. Gyakran adódik olyan helyzet is, amikor különböző kényesnek számító műveleteket nem hajthat végre egy időben mindkét fél, ezért a viszonyrétegnek egy vezérjelről kell gondoskodnia. A vezérjel egy időben mindig csak az egyik félnél lehet, és csak akkor hajthat végre bizonyos műveleteket, ha a vezérjel éppen nála van. Másrészt a viszonyrétegnek az összehangolás is a feladatai közé tartozik: a sikeres kapcsolathoz elengedhetetlen, hogy a két fél egymással összhangban legyen. Az összehangolás lehetővé teszi, hogy a kapcsolat megszakadásakor se kelljen mindent elölről kezdeni. Ha például egy fájlátvitel megszakad, akkor a kapcsolat újrafelépítésével az átvitelt onnan folytathassuk, ahol az félbemaradt.

A megjelenítési réteg a felhasználók életét hivatott megkönnyíteni. Célja, hogy a minden alkalommal (vagy legalábbis gyakran) elvégzendő feladatokat a felhasználók helyett megvalósítsa; ilyesmi például az alsó rétegek számára továbbítandó adatok átalakítása. Erre azért van szükség, mert a felhasználói programok elég bonyolult adatszerkezetekkel dolgoznak, s ezek egy az egyben nem vihetőek rá a kommunikációs csatornára. A másik nem elhanyagolható dolog, hogy minden rendszer más és más kódot használ a karaktersorozatok megjelenítésére – az ebből adódó különbségeket is le kell küzdeni.

Az utolsó és egyben legfelső réteg az alkalmazási réteg. Valódi feladata az, hogy a különböző operációs rendszerek együttműködési képtelenségéből adódó nehézségeket kiküszöbölje. Gondoljunk csak a fájlátvitelre: minden operációs rendszer másként tárolja az állományokat. Egy hálózaton ugyanakkor elvárható, hogy például egy linuxos gépről egy VMS rendszerre is küldhessünk állományokat. Az alkalmazási réteghez még számos más protokoll tartozik, amelyek például a felhasználók közötti levelezést és egyéb különlegesebb feladatokat látnak el.

Gondok az OSI modellel

Sorozatunkat ugyan az OSI modell rétegeire fogjuk építeni, mindazonáltal az OSI modellel csupán annyi gond van, hogy gyakorlati alkalmazása szinte lehetetlen. Valójában még senkinek sem ment az OSI modell teljesen sikeres

megvalósítása. Ennek okai a következők:

- Először is maga a módszer rossz. Gond van a rétegek elrendezésével. A viszony és a megjelenítési réteg szinte teljesen üres (bizonyos hálózatokban el is hagyják), az adatkapcsolati és a hálózati rétegnek viszont túl sok a feladata. Ezeket talán jobb lett volna további rétegekre bontani. Az OSI egyébként eredetileg öt rétegből (a viszony- és a megjelenítési réteg nélkül) állt. A rétegek számának hétre történő módosítása az IBM-től való félelemnek tudható be. Az IBM is készített egy saját hétrétegű hivatkozási modellt, amelynek a neve nemes egyszerűséggel System Network Architecture (SNA) volt. Akkoriban az IBM eléggé befolyásos cég volt, és sokan tartottak tőle, hogy erőfölényét kihasználva mindenkire rákényszeríti az SNA-t. Mivel ez az IBM saját szabványa volt, azt bármikor kedve szerint változtatgathatta volna. Ezért volt szükség az OSI-ra mint egy olyan – szintén hétrétegű – modellre, amelyre egy nemzetközi bizottság felügyel.
- A másik gond az OSI-val, hogy rendkívül bonyolult és számos érthetetlen módosítás került bele az évek során: ilyenek voltak például az alsó rétegekbe beágyazott hibajavítási és címzési szolgáltatások. Sokan rámutattak, hogyha ezeket a szolgáltatásokat a felsőbb rétegek végzik, akkor nagyobb hatékonyság érhető el. Bizonyos szolgáltatásokat pedig egyáltalán nem tartalmaz a modell, például a titkosítást vagy a hálózatirányítást.
- A legeslegsúlyosabb gond azonban az OSI mögötti szemléletmód, amelynek alapján az egészet kitalálták. A gond a következő: kapcsolatot a `CONNECT.request` primitívvel lehet létrehozni. Ehhez egy rendszerhívásnak (vagy egy osztott könyvtárban elhelyezett függvénynek) kell tartoznia, amelynek segítségével egy folyamat egy másik gépen futó folyamattal kapcsolatba tud lépni. A modell szerint ilyenkor a `CONNECT.indication` nevű primitív jelez a vevőfolyamatnak, hogy valaki kapcsolatba akar lépni vele. Ezt megszakítás segítségével érhetjük el. A hálózati alkalmazásokat azonban magas szintű programozási nyelvekben írják, és ott rendkívül kellemetlen dolog a megszakításkezelés. Sokkal jobb megoldás, ha létezik mondjuk egy `receive()` nevű rendszerhívás, amelyik a folyamatot egészen addig blokkolja, amíg valaki kapcsolatot nem kezdeményez vele. Ezzel teljes egészében kiválthatnánk a `CONNECT.indication` primitívet.

Amiért mégsem így van, az annak tudható be, hogy az OSI modell tervezésekor a kommunikáció elvét követték. Ez valami olyasmi, mint a telefon. Amikor keresnek minket, a telefon csörögni kezd. Amikor egy számítógépen egy folyamatot keresünk, akkor az nem képes csörögni, legfeljebb megszakításokat idézhet elő. Ez a szemlélet azonban gyökeresen ellentétben áll a korszerű programozási elvekkel. Volt más dolog is, ami hozzájárul az OSI modell bukásához: megjelent a Berkeley Unix nevű operációs rendszer, amely egy ingyenes és viszonylag jól működő, az OSI-hoz hasonló hivatkozásimodell-megvalósítást tartalmazott. Ez a modell pedig a TCP/IP volt. A következő részben innentől folytatjuk.

Garzó András (garzo@elte.hu)