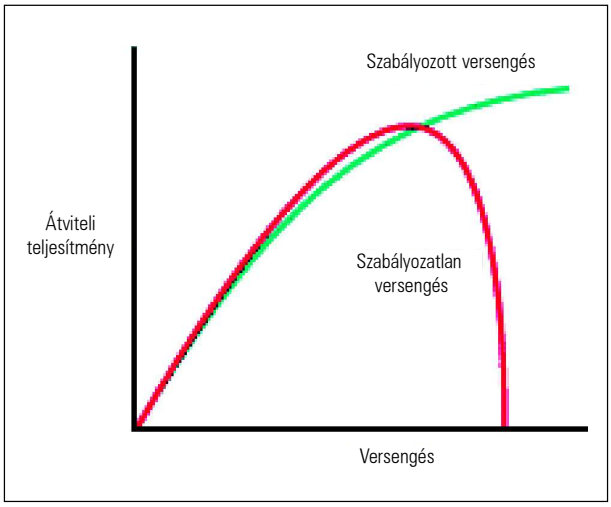
Számítógép-hálózatok (18. rész)
Torlódásvédelem

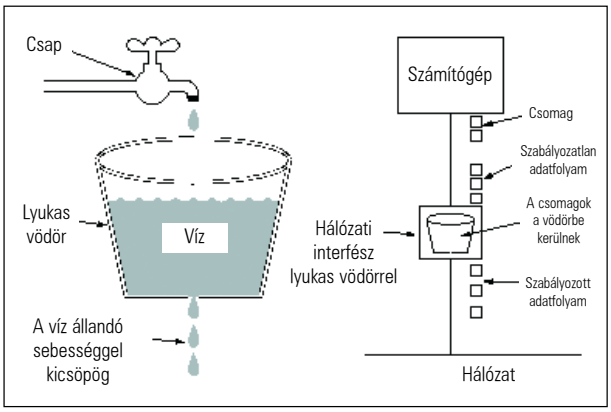
A torlódások elleni védekezés nélkül előbb-utóbb egyetlen csomagot sem tudunk átvinni a hálózaton. Éppen ezért a torlódásvédelem a hálózati réteg egyik legfontosabb feladata, és egyben a sorozat jelen részének témája.

Az előző részből megtudhattuk, hogy a hálózatunkra leselkedő legfélelmetesebb veszély a torlódás kialakulása. Torlódást sok minden előidézhet, és ráadásul saját magát gerjeszti, azaz ha egyszer valahol kialakul, akkor rövid úton átterjed az alhálózat más pontjaira is. Az *1. ábrán* láthatjuk, milyen következményekkel járhat az, ha a torlódások ellen nem lépünk fel kellő eréllyel. Itt a kézbesített csomagokat ábrázoltuk az elküldött csomagok függvényében. Amikor még az alhálózatnak átadott csomagok száma az átviteli kapacitáson belül marad, addig minden rendben zajlik, a csomagok csak átviteli hiba következtében veszhetnek el. (Tehát az elküldött és a kézbesített csomagok egyenes arányban vannak egymással).
Ha viszont egyszerre túl sok csomaggal árasztjuk el szerencsétlen alhálózatunkat, akkor az útválasztók puffer-e előbb-utóbb megtelik, így bizonyos csomagok el fognak veszni. Ez azonban tovább súlyosbítja a helyzetet, ugyanis a gépek az elveszett csomagot ismét megpróbálják elküldeni, ezzel is növelve a már amúgy is leterhelt alhálózat forgalmát. A helyzet egészen odáig fajulhat, hogy egyetlen csomagot sem leszünk képesek átküldeni az alhálózaton. A torlódásvédelem tehát a hálózati réteg egyik legfontosabb feladata.
A torlódásvédelemnek tehát biztosítani kell azt, hogy az alhálózat képes legyen megbirkózni a legkülönfélébb forgalmi helyzetekkel, és gond nélkül továbbíthassa a rajta áthaladó csomagokat. A legelső dolog, amit a torlódásvédelemmel kapcsolatban tudnunk kell az, hogy ez egy globális, az egész hálózatot átszövő kérdés. Amikor megpróbálunk védekezni a torlódások ellen, nem elég, ha például csak az útválasztók csomagküldési mechanizmusán próbálunk valamit állítgatni. Minden olyan tényezőt figyelembe kell vennünk, amely hatással lehet az alhálózaton jelenlévő forgalomra, mint például a gépek viselkedése. Az eredményes védekezés érdekében tehát a hálózat összes alkotóelemének együtt kell működnie. Így egy torlódás kialakulásakor a gépeknek is meg kell tenniük a megfelelő lépéseket, például egy ideig lelassítani, vagy esetleg teljesen felfüggeszteni a csomagok küldését.



1. ábra

Fontos, hogy ne keverjük össze a forgalomszabályozás és a torlódásvédelem fogalmát. Igaz, hogy az előbbi esetben is szükség van arra, hogy a gépek időnként képességeikhez mérten lassabban adjanak, a két fogalom mégis merőben más. A forgalomszabályozás csak az adóval és a vevővel, és a kettőjük között kiépült kétpontos kapcsolatban kialakuló torlódásokkal foglalkozik.
Forgalomszabályozásra például akkor lehet szükség, amikor egy processzorokkal jól megpakolt szuperszámítógép küld adatokat a kis személyi számítógépünknek mondjuk egy 1 Gb/s-os sávszélességű vonalon. A szuperszámítógépnek nem okoz gondot ekkorra sebességgel adni, a kisebb kapacitású gépünk képességeit ez a sebesség azonban meghaladja. A gyorsabb gépnek tehát vissza kell vennie a sebességből, pedig magán a hálózaton nem lépett fel torlódás.
Ezzel ellentétben a torlódásvédelem a hálózaton megjelenő torlódásokkal foglalkozik. Például amikor egyik LAN-ról a másik LAN-ra egyszerre sok állományt szeretnénk átküldeni. A LAN-ban lévő gépek ezzel a feladattal simán



2. ábra

megbirkóznának, az alhálózat azonban beledöglene, hiszen az átküldendő csomagok száma meghaladja az áteresztőképességét. A baj elkerüléséhez a gépeket rá kell bírni arra, hogy ne egyszerre árasszák el az alhálózatot, az útválasztóknak pedig a forgalmat szét kell osztaniuk több útvonal között.

A torlódásvédelmi algoritmusok durván két nagy csoportra oszthatók: a nyílt hurkúakra ***(open loop)*** és a zárt hurkúakra ***(closed loop)***. Az előbbi osztályba tartozó módszerek úgy próbálják a torlódásokat elkerülni, hogy eleve nem hagyják azokat kialakulni. Ehhez különböző irányelveket határoznak meg, amelyek alapján az útválasztók és a gépek döntéseiket meghozzák. Fontos, hogy ezek kőbevésett irányelvek, amik örökérvényűek, azaz nem változnak a hálózat működése közben. Ez azt is jelenti, hogy a gépek és az útválasztók döntéseikben nem veszik figyelembe azt, hogy éppen mi is történik a hálózaton (azaz mi a hálózat aktuális forgalmi állapota).

Ezzel gyökeresen ellentétes felfogást képviselnek a zárt hurkú algoritmusok, amelyek csak akkor avatkoznak be, amikor már valahol kialakult a torlódás. Ilyenkor gyors információgyűjtésbe kezdenek, amelyből megállapítják, milyen lépéseket kell tenni a torlódás megszüntetésének érdekében. Ezután felszólítják a megfelelő útválasztókat, illetve gépeket, hogy tegyék meg a megfelelő intézkedéseket.

A következőkben mindkét típusú algoritmusra mutatunk példákat, és megvizsgáljuk, miként valósítják meg a gyakorlatban a fent leírtakat.

Nyílt hurkú torlódásvédelmi algoritmusok

A nyílt hurkú eljárások tehát arra törekednek, hogy esély se legyen a torlódás kialakulására, ehhez pedig meg kell szüntetni a torlódás legfőbb okát, a lökésszerű forgalmat. A gépek általában még véletlenül sem adnak egyenletesen, hanem mindig a legváratlanabb pillanatban halmozzák el az alhálózatot csomagokkal. Ha a gépeket rábírhatnánk, hogy egyenletesebben küldjék csomagjaikat, akkor kisebb lenne az esély a torlódás kialakulására.

Ezzel el is érkeztünk a ***forgalomformálás (traffic shaping)*** fogalmához, amely az alhálózaton folyó adatátvitel ***átlagos*** sebességének szabályozását jelenti. Most megnézünk két olyan algoritmust, amelynek segítségével ezt a sebességet kordában tarthatjuk.

Ezek közül az első a ***lyukas vödör (leaky bucket)*** algoritmus, amelynek nagyon találó neve van, hiszen tényleg úgy működik, mint egy valódi vödör, amelynek az alján lyuk tátong *(2. ábra)*. Amikor egy ilyen vödörbe vizet eresztünk, az alján a víz állandó sebességgel fog szivárogni, mégpedig úgy, hogy ez a sebesség nem függ attól, milyen gyorsan eresztjük a vizet a vödörbe.

Most képzeljük el ugyanezt, csak víz helyet csomagokat lapátolunk a lyukas vödrünkbe, amit a gépek egy sorként valósítanak meg. A sorba folyamatosan érkezhetnek a csomagok, egészen addig, amíg meg nem telik.

Ha egy csomag nem fér be a sorba, akkor az eldobásra kerül. Minden óraütéskor egy csomag kikerülhet a sorból, és elküldhető az alhálózatnak. Ezzel a módszerrel tehát megkímélhetjük az alhálózatot a löketszerű forgalomtól.

Mivel a csomagok egyenletesen továbbítódnak, az útválasztóknak több idejük van egy-egy csomag feldolgozására, nehezebben telik meg a pufferük, így csökken a torlódás kialakulásának esélye.

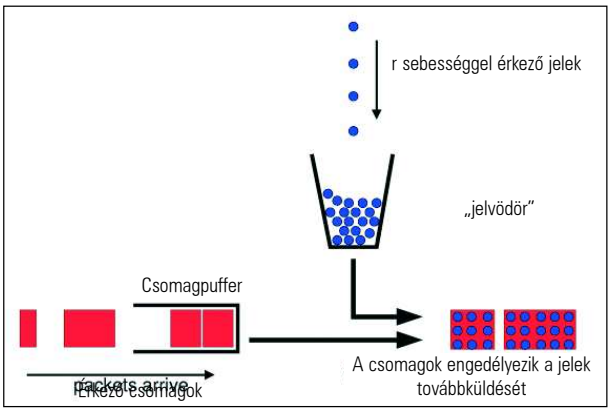
Ez a módszer azonban nem minden esetben tökéletes. Hibája abban rejlik, hogy sziklaszilárdan ragaszkodik az általa kialakított átlagos csomagküldési sebességhez (más szóval kimeneti mintához). Bizonyos esetekben azonban szükség lehet egy kis rugalmasságra. Néhány alkalmazás ugyanis úgy működik, hogy sokáig számolgat ***(vagy épp semmit sem csinál)***, majd hirtelen egy nagyobb löketet indít útnak, majd jó ideig megint nem küld semmit.

Ha a gép sokáig csendben volt, akkor nem nagy baj, ha egy kicsit nagyobb kimeneti sebességgel kezd adni, majd utána fokozatosan visszavesz belőle. Ha ilyenkor is ragaszkodunk a lyukas vödör merev kimeneti mintájához, akkor a csomagok lassabban jutnak célba. Probléma merül fel akkor is, ha az alkalmazás több csomagot akar küldeni, mint amennyi a vödörbe belefér, hiszen ilyenkor az utolsó csomagok elvesznek.

Ezekre a problémákra kínál megoldást a ***vezérjeles vödör (token bucket)*** algoritmus *(3. ábra)*. Itt a vödörben nem csomagokat, hanem úgynevezett vezérjeleket tárolunk. A vezérjelek óraütésenként, folyamatosan keletkeznek. Amikor a vödör megtelik vezérjelekkel, akkor az újonnan születő vezérjelek megsemmisülnek. A gépet egy csomag csak akkor hagyhatja el, ha kivesz egy vezérjelet a vödörből. Ha a vödör kiürült, akkor addig kell várnia, amíg új vezérjel nem keletkezik.

Ez az algoritmus két jelentős dologban is eltér a lyukas vödörtől. Először is itt a gépek némileg nagyobb önállósággal bírnak, hiszen spórolhatnak a verzérjelekkel. Azok a gépek, akik egy ideig nem adnak, félretehetik vezérjeleiket egy nagyobb löket számára. Persze ez a löket nem lehet nagyobb, mint a vödör mérete. Másik fontos különbség, hogy a vödör megtelése után itt csak vezérjeleket dobunk el, csomagokat azonban soha.

Érdemes megjegyeznünk, hogy e két algoritmust nem csak a gépek kimeneti mintáinak szabályozására használják, hanem például két útválasztó közötti forgalomkisimításra is. Ilyenkor azonban kell némi változtatás. Gépek esetében a vezérjeles vödör használatakor nem jelent túl nagy problémát, ha nem engedünk ki újabb csomagot addig, amíg új vezérjel nem keletkezik. Útválasztók eseté-



3. ábra

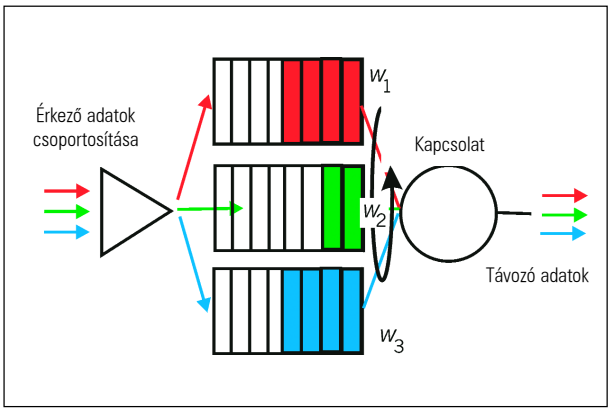
ben a csap nem mindig zárható el. Tegyük fel például, hogy a vezérjelek elfogynak, de továbbra is érkeznek befelé jövő csomagok. Ha ilyenkor korlátozzuk a kimenetet, akkor fennáll a veszély, hogy a puffer megtelik, és egyes csomagok elvesznek.

Zárt hurkú torlódásvédelem

A torlódásvédelmi algoritmusok másik nagy csoportja nem a torlódásokat dinamikusan próbálja szabályozni, azaz folyamatosan felügyeli a hálózat aktuális forgalmi állapotát, és ahol és amikor szükségét látja, beavatkozik.
Felmerülhet a kérdés, hogy a hálózaton kialakuló torlódások nagyságát milyen mennyiséggel tudjuk jellemezni. Erre sok választási lehetőség adódik. Figyelhetjük például az útválasztók átlagos sorhosszait (egy útválasztó pufferjében átlagosan hány darab csomag vár továbbításra), az újraküldött csomagok számát, vagy éppen az útválasztók puffertúlcsordulásából adódó csomagvesztések arányát. Bármelyik paramétert figyelje is ezek közül egy algoritmus, annyiban biztosak lehetünk, hogy minél magasabb értéket mér, annál súlyosabb a helyzet a hálózaton.
Amikor egy útválasztó torlódást észlel, akkor azt azonnal jelezni kell társainak, mivel csak együttes fellépéssel fékezhetik meg a kialakult forgalmi dugót. Erre is sok módszer létezik. Ezek közül a legkézenfekvőbb az, ha speciális csomagokat küldünk szét, amelyből a többi útválasztó értesül a kialakult helyzetről. Ezzel azonban az a baj, hogy az amúgy is már terhelt hálózatot tovább lassítanánk. Így sokkal praktikusabb megoldásnak ígérkezik az, ha minden csomag fejlécében fenntartunk egy bitet, és annak segítségével figyelmezteti a többieket. Ha a terhelés meghalad egy bizonyos küszöböt, akkor minden kimenő csomagnál az útválasztó 1-esre állítja ezt a bitet, ezzel jelezve a többieknek, hogy vegyenek vissza az adási sebességből.
Másik megoldás lehet a forgalom megosztása. Ilyenkor az útválasztók folyamatosan mérik a vonalak terheltségét, és úgy irányítják a csomagokat, hogy azok lehetőleg teljesen elkerüljék az épp torlódás alatt álló részt.
Most nézzünk meg egy-két konkrét algoritmust! Virtuális áramkör alapú alhálózatok esetében (ilyen például a telefonhálózat) a legegyszerűbb megoldás a ***belépéses ellenőrzés (admission control)***. A módszer azon az egyszerű feltételezésen alapul, hogyha valahol torlódás alakult ki, akkor a helyzeten valószínűleg nem fog segíteni az, ha engedünk további virtuális áramkörök ***(kapcsolatok)*** kiépítését. A vonalas telefonhálózatoknál is hasonló a helyzet: a tárcsahangot egészen addig nem kapjuk meg, amíg a hálózat fel nem szabadul a terhelés alól.

Lefojtó csomagok

Ezen az eljáráson sok torlódásvédelmi algoritmus alapul, és alkalmazható mind datagram, mind virtuális áramkör alapú alhálózatok esetében is.
Az alapötlet itt is az, hogy az útválasztók folyamatosan figyelik a kimeneti vonalaikat, és mindegyikhez, terhelésüktől függően hozzárendelnek egy 0 és 1 közé eső valós számot. (Egyes algoritmusok az útválasztó más erőforrásait kísérik figyelemmel, például a puffer méretét). Ha ez a szám meghalad valamiféle küszöbértéket, akkor az útválasztó minden befelé menő csomag forrásához visszaküld egy úgynevezett ***lefojtó csomagot (choke packet)***. Amikor a gép egy ilyen csomagot kap, akkor azonnal visszavesz az adási sebességéből (például a lyukas vödör algoritmus segítségével). A lefojtó csomagok ugyan „meg vannak jelölve", s így az útválasztók nem generálnak továbbiakat, de mivel egy torlódás általában több útválasztót is érint, a gép egyszerre több lefojtó csomagot is fog kapni. Ilyenkor a sebesség visszavétele után egy ideig nem fog reagálni az lefojtó csomagokra.
Ezután a gép várakozik és figyel. Ha továbbra is érkeznek lefojtó csomagok, akkor ez azt jelenti, hogy a torlódás még mindig fennáll, így az adási sebességet tovább csökkenti. Ha azonban bizonyos idő elteltével nem érkezik újabb elfojtó csomag, akkor a torlódás valószínűleg megszűnt, így nem kell már korlátozni a forgalmat.
Nem egyszerű kérdés, hogy a gép mennyi idő múlva adja fel a sebességkorlátozást. Ha túl hamar teszi, akkor nagy rá az esély, hogy újabb torlódást idéz elő, ha pedig túl lassan, akkor meg nem működik gazdaságosan. A megoldás minden bizonnyal a két véglet között lesz, de hogy pontosan hol, arra csak az adott algoritmus pontos elemzése adhat választ. Hasonló megfontolásokat kíván annak megállapítása is, hogy az algoritmus a lefojtó csomagok hatására milyen ütemben csökkentse sebességét, illetve a torlódás megszűnésével miként növelje azt. A jól bevált módszer az, ha torlódás esetén a gépek megfelezik adási sebességüket, viszont utána csak kisebb lépésekben növelik azt.
Ennek az algoritmusnak van viszont egy olyan hátránya, hogy feltétel nélkül hisz a gépek becsületességében. A hálózatok világa azonban igazságtalan. Ha egy gép kötelességtudóan lecsökkenti adási sebességét, de a többiek ezt nem teszik meg, akkor a becsületes gép jár a legrosszabbul, mivel neki drasztikusan le fog csökkenni a sávszélessége.
Egy igazságosabb világ ígéretét rejti magában a ***pártatlan sorbaállítás (fair queueing)*** algoritmus. Itt az útválasztók minden kimeneti vonalhoz pontosan annyi várakozási sort rendelnek, ahány bemenetük van. A várakozási sorokon egyenként, körkörösen végigmegyünk, és kiveszünk egy csomagot, amit ráteszünk a kimenő vonalra. Ennek köszönhetően az egymással versengő gépek egyenlő mennyiségű csomagot tudnak célba juttatni.



4. ábra

A dzsitter szabályozás

Minden felhasználó azt szeretné, ha a neki szánt csomagok a lehető leggyosabban elérnének hozzá, azaz a letöltési sebessége minél nagyobb legyen. A felhasználók tehát rendkívül rosszul tűrik, ha a csomagjaik késnek, például azért, mert egyes útválasztók feltartóztatják azokat.

Egyes alkalmazások azonban nem a csomagkésleltetésre, hanem inkább az úgynevezett dzsittere érzékenyek, azaz a csomagok (a forrástól a célig történő) átviteli idők durva változásaira. Például a hang, illetve mozgókép továbbításkor simán belefér, ha minden csomag csúszik pár századmásodpercet, viszont akkor előbb se jöjjön senki, mindenki ugyanannyit késsen.

Ehhez először meg kell becsülni a csomagok várható átviteli idejét (beleszámolva az átlagos torlódást is), ugrásokra lebontva. Így minden útválasztó tudni fogja, hogy optimális esetben mekkora időközönként kell egyegy csomagot továbbítania. Amikor egy ilyen csomag érkezik az útválasztóhoz, megnézi, hogy nincs-e késésben, illetve nem jött-e előbb a kelleténél. Az utóbbi esetben nincs már dolga, mint hogy kivárja a megfelelő időt, majd útjára bocsátja. Ha siet, akkor nem tehet mást, minthogy elsőbbséget ad neki az összes többi csomaggal szemben, és reménykedik, hogy így még be tudja hozni a késését, és a dzsitter nagysága nem nő meg számottevő mértékben.

Terhelés eltávolítása (load schedding)

Amikor már nagyon veszélyes a helyzet, és a csomagok az útválasztók fejére nőttek, akkor jön a végső megoldás: az útválasztók se szó, se beszéd, elkezdik kidobálni a csomagokat. Na de milyen elv alapján döntsük el, mely csomagok kerüljenek végérvényesen a süllyesztőbe? Nem hangzik túl jó ötletnek, ha véletlen szerűen, például minden második csomagnak kegyelmeznénk meg, a többit kérdés nélkül kidobnánk.

Sokkal jobb lenne, ha ezt a csomagot küldő alkalmazástól tennénk függővé, ugyanis mindegyiknek más és más csomag a legfontosabb. Mit is értünk ezalatt? Egy fájlküldő programnak, például egy FTP kliensnek inkább a korábban jött csomagok a fontosak, mert ha például egy vagy több csomag kimarad, akkor elképzelhető, hogy a forrásnak a sikeresen megkapott csomagokat is újra meg kell ismételnie. A ***RealPlayer*** azonban, ha választhatna, inkább a legújabb csomagokat kapná meg.

A legjobb megoldás tehát az, ha maga az alkalmazások mondhassák meg, mely csomagok a legfontosabbak számukra. Ehhez lehetőséget kell nekik biztosítani arra, hogy a csomagok fejlécébe jelöljék, az adott csomag mennyire fontos. Magyarul prioritásokat rendelhetünk a csomagokhoz, és ha torlódás van, akkor az útválasztók először a legalacsonyabb prioritási osztályba sorolt csomagoktól válnak meg.

Ahhoz, hogy ez a módszer jól működjön, figyelembe kell vennünk egy nagyon is emberi tényezőt, mégpedig azt, hogy az emberek maguktól valószínűleg nem fognak alacsonyabb prioritású csomagokat küldeni. Ha nincs semmifajta kényszerítő erő, nyílván mindenki a legmagasabb prioritásra fogja állítani a csomagjait. A megoldás erre lehetne például az, hogy a szolgáltatók drágább tarifát számítanak fel a magasabb prioritású csomagokért.

Bizonyos hálózatokban azonban nem kecsegtet akkora haszonnal az, ha egyes csomagokat magasabb prioritással láthatunk el. Az ATM esetében például a csomagok fix méretű cellák, és ezek egy nagyobb egység részei.

Egy üzenet több fix méretű cellaként halad tovább, és ha az útválasztónak egy cellát el kell dobnia, akkor valószínűleg a forrásnak az egész üzenetet újra meg kell ismételnie. (Mindezek ellenére az ATM cellák fejlécében van egy bit, amellyel megkülönböztethetjük a fontosabb csomagokat a többiektől).

A végére még egy megjegyzéssel élnénk. A szimulációs tesztek azt mutatják, hogy az útválasztó, és vele együtt az alhálózat akkor jár a legjobban, ha a torlódás észlelése után minél előbb nekiáll a csomagok pusztításának, mivel így elejét vehetik egy komolyabb forgalmi dugó kialakulásának. Tehát megint egy újabb dilemma. Inkább minél több csomagot próbáljunk célba juttatni, és lefojtó csomagok, illetve egyéb technikákkal próbáljuk csillapítani a kialakuló torlódásokat, vagy a torlódásvédelem minden felett álljon, és a baj legkisebb jelére már pusztítsuk a csomagokat.

Nos, ennyit a torlódásvédelemről. A sorozat következő részei sokkal életszerűbbek lesznek, mostantól ugyanis elvetjük azt a feltételezésünket, hogy minden hálózat egyforma. Eddig vígan küldözgettünk csomagokat hálózatok között, ám fel sem merültek bennünk olyan alapkérdések, mint például miként tudnak együttműködni eltérő protokollokat használó hálózatok.

A hálózatok persze nem csak protokolljaikban különböznek egymástól, hanem még vagy száz másik dologban. Ezek között bizony átjárást kell biztosítani, amelyekről az ***átjáró (gateway)*** nevű eszközök fognak gondoskodni. Hogy pontosan miképp, az az elkövetkezendő pár rész témája lesz.

Garzó András (garzoand@interware.hu)
Körülbelül három éve foglalkozik Linux- és más Unix-rendszerekkel. Legjobban az operációs rendszerek lelkivilága érdekli, de nyitott egyéniség. Kedvenc étele a palacsinta, és van egy Richard nevű macskája. Minden észrevételt, megjegyzést, levelet szívesen fogad.