

Hálózati sávszélesség-menedzsment

Mátó Péter

<mato.peter@andrews.hu>

Zámbó Marcell

<zambo.marcell@andrews.hu>

Kivonat

Ki ne ismerné az érzést, mikor egy letöltés annyira leterheli a hálózatot, hogy semmi mást nem lehet mellette csinálni? Állandó nyűg ez kis és nagy hálózaton egyaránt. Az otthoni gépen szaggat az IP telefon, mert a gyerek a saját gépén éppen hálózaton játszik, és megeszi a családi ADSL vonal sávszélességét? A céges hálózaton nem tudnak közlekedni a fontos üzleti levelek, mert túl sok felhasználó akar egyszerre autós filmeket letölteni?

Nem túl közismert, de van megoldás. És nem is ördögösség. A Linux rendszermagnak mindig is erőssége volt a modern hálózati technológiák támogatása. Nagyon régóta elérhető többféle sávszélesség-menedzsment rendszer, melyek segítségével és némi hozzáértéssel pontosan szabályozhatjuk a gép hálózati kártyáin áramló adatforgalmat. Az előadásból a hallgatóság megismerheti az interneten használt szállító protokollok néhány, a sávszélesség szabályozásának szempontjából fontos sajátosságát. Egyszerű példákon keresztül bemutatjuk, hogyan lehet egy asztali gépen bizonyos hálózati forgalmakat behatárolni, végül az előadás utolsó részében egy összetettebb, tűzfalakon vagy otthoni kijáraton is használható megoldást mutatunk be.

A fontosabb címszavak: hálózat, sávszélesség, sávszélesség-menedzsment, TCP, TCP ablak, ICMP, hálózati forgalom, csomagszűrő, hálózati csomag, sávszélesség osztályok, hálózati csatoló, várakozási sorok, csomag osztályzás, priorítás, sorba állítási módszerek, mangle tábla, szabályrendszer, tűzfal, iptables, súlyozás. . .

És amit nemigen lehet magyarra fordítani: ingress, egress, burst, mark, routing, router, throughput, overhead. . .

Tartalomjegyzék

1. Bevezető	3
2. Hálózati kapcsolatok	3
3. Protokollok	4
3.1. Az ICMP és UDP jellemzői	5
3.2. A TCP jellemzői	5
3.3. A legfontosabb hálózati protokollok sávszélesség jellemzői	7
4. A sávszélesség-menedzsmentről	8
4.1. A forgalom szabályzásának módszerei	8
5. A Linux TC alapjai	9
5.1. A Linux TC kernel részei	10
5.2. Besorolási módszerek (qdisc)	11
5.3. Osztálymentes qdisc-ek (classless)	11
5.4. Osztály alapú qdisc-ek (classful)	12
5.5. A forgalom osztályozása	13
6. A kimenő forgalom szabályozása	14
7. A bejövő forgalom szabályozása	15

1. Bevezető

Köszönet a sok okosságért Bozónak, a TCP varázslónak.

Sávszélesség. A wikipédia [1] szerint: egy digitális kapcsolaton adott idő alatt átvihető adat mennyisége. Gyakran bitsebességnek is nevezik. Mértékegysége a bit/sec vagy byte/sec, ami az másodpercenként átvihető bitek vagy bájtok számát adja meg. Napjainkban a hálózatok sebességének növekedése miatt a kbit/sec vagy inkább a Mbit/sec az elterjedt. Sávszélesség. Ma, a hálózatok korában már majd mindenkinek van, de senkinek nincs elég (minden sávszélességet ki lehet tölteni, mint ahogy tárhelyből sincs soha túl sok). Akinek kevés van, az jól szeretne gazdálkodni vele, akinek sok van, az szeretné okosan elosztani. Hogyan lehet ezt egyszerűen megoldani? Erről szól ez az írás.

2. Hálózati kapcsolatok

Kezdetben volt a telefonos hálózat. Akkoriban a kiválasztottak modemeket használtak, amelyet az egyszerű népek csak csodálhattak. Ebben az időben a sebesség még csak 2400bit/sec, de ez elsősorban a telefonrendszerek rossz minőségének volt köszönhető. Kalandos idők voltak. Ahogy javultak a vonalak, a modemekkel akár 56 kbit/sec sebességet is el lehetett érni – ilyen kapcsolatokat még ma is lehet találni ott, ahol a telefonközpontok vagy a kábeltévé hálózatok nincsenek megfelelően kiépítve. A modemek le- és feltöltési sebessége általában megegyezik (az 56k-s modem esetén 33.6k a feltöltési sebesség), a modem típusától függően vagy párhuzamos a fel- és a letöltés (full-duplex), akár a beszéd a telefonvonalon, vagy egyszerre csak az egyik fél küldhet adatot (half-duplex), mint a CB rádió esetében. Előkerült egy nagyon fontos fogalom-pár: a fel- és letöltés. Később meglátjuk, hogy ezek a fogalmak nagyon fontosak a további történetünk szempontjából, ezért beszéljünk egy kicsit róluk.

Valamiért az kép alakult ki az emberek fejében, hogy a kliensszámítógép fizikailag és virtuálisan alacsonyabb szinten helyezkedik el, mint a szerver. Ezért letöltésnek hívják, amikor a szerver adatot küld a kliensnek (például egy munkaállomásnak), és feltöltésnek ha a kliens továbbítja adatot a szervernek. Gyakran hallhatjuk például, hogy letöltöttem egy remek albumot az internetről, vagy hogy feltöltöttem az új weblapot a szerverre. A sávszélesség-menedzsment szempontjából a két irány egészen más tulajdonságokkal bír, de erről majd később.

Az analóg modemet szerencsére lassan már mindenhol le lehet vinni a pincébe, mert végre elérkezett a digitális kor. Nálunk a modem után a leginkább elterjedt technológia az ISDN (Integrated Services Digital Network), amely lényegesen nagyobb sávszélességet tesz lehetővé (kb. 128 kbit/s, ha mind a két csatornát használják), de a telefonszámlája miatt anyagilag nem túl barátságos, ezért nem is szeretjük. Jelenleg leginkább az ADSL, WLAN és kábelt megoldásokat kedvelik a felhasználók. Az ADSL (Asymmetric Digital Subscriber Line) ma már nagyon sok helyen hozzáférhető (igaz, főleg a városokban), a sávszélessége nagyon jó: jelenleg nálunk legfeljebb 4Mbit/s le, 512 kbit/s fel. Itt már látszik, hogy miért emeltük ki korábban a le és feltöltés különbségét. De ez csak a kezdet. Az egyszerű emberek, akik nem üzemeltetnek webszervert az otthoni számítógépükön, lényegesen többet töltenek le, mint fel, ezért ez a megoldás nekik tökéletesen megfelel. Annyi gond van vele, hogy ha a felfelé irányuló csatorna bedugul, akkor bizony lefelé se nagyon tudnak haladni az adatok. Ennek okáról később, a protokollok tárgyalásánál írunk.

A WLAN és a kábelt technikailag nagy sávszélességű, szimmetrikus adatátvitelt tesz lehetővé, de általában itt is korlátozzák a feltöltési sebességet. Ennek az az oka, hogy a szolgáltatók nem vesznek felesleges sávszélességet, ezért nem szeretnék, ha a felhasználók otthon hobbiszervereket üzemeltetnének.

Szinte minden kapcsolatfajtánál előfordulhat, hogy ha bizonyos forgalmak túlzottan rátelepszene a sávszélességre, akkor mások nem tudnak kényelmesen működni. Gyakori eset, mikor néhány nagyobb letöltés (mert kijött az új Debian stabil, és azonnal le kell tölteni a CD-ket) annyira leterheli a vonalat, hogy az internetes telefon nem működik megfelelően. Itt válasszuk szét a hálózati forgalmat két típusra. Az interaktív forgalom általában kicsi, de azonnal továbbítani kell. Ilyenek a már említett IP telefon és videó, az interaktív távoli hozzáférési módszerek (telnet, ssh, vnc...), a csevegő alkalmazások (IRC, webchat, ICQ stb.) és végül is ide sorolható a kisebb weboldalak átvitele is. Ez esetekben nagyon fontos az adatok azonnali átvitele, mert egyébként szaggat a hang, nem jönnek le a lapok és az nem jó. A hálózati forgalmak másik típusa a tömeges adatok átvitele (bulk transfer), ahol az azonnaliság nem annyira fontos szempont. Ilyenek a nagyobb letöltések (web és ftp egyaránt), a levelek küldése és fogadása, rendszerfrissítés és így tovább. Meg kell ismerkednünk két új fogalommal: ezek a throughput és a látencia (latency). A throughput a vonal átviteli sebessége, a látencia pedig a vonal késleltetése.

Az interaktív forgalmakat érdemes lenne egy külön, kicsi VIP csatornán átvinni, ahol a késleltetést igyekeznénk minimalizálni, míg a masszív adatátvitelnél olyan csatorna kellene, ahol az átvitelt maximalizálnánk. De nekünk csak egy vonalunk van, az is bizonyos korlátokkal rendelkezik technológiai sajátosságai miatt. Hogyan lehet mégis használható kompromisszumos megoldást kialakítani? Lássuk!

3. Protokollok

A továbbiak megértéséhez sajnos le kell szállnunk bit szintre, és meg kell vizsgálnunk az interneten leggyakrabban használt protokollok sajátosságait. Aki viszolyog a hexa editoroktól, annak javasoljuk, hogy a következő bekezdés elolvasása után ugorjon a következő fejezetre.

Vezetői összefoglaló a protokollokról: az IP protokoll adatok átvitelére való számítógépek között, egy forrás- és egy cél cím található benne. Ezek az ún. IP címek azonosítják a számítógépeket az interneten. Az IP protokoll nem csinál mást, mint a hátán cipel más protokollokat a címzetteknek és vissza; leggyakrabban TCP és UDP protokollokat szállít. Amik valójában nem is tudják, hogy honnan, hová mennek, csak annyit, hogy ha célhoz értek, akkor melyik szolgáltatásnak vannak címezve. A szolgáltatásokat ezekben a protokollokban a portok címzik meg. Az UDP sajátossága, hogy nagy megbízhatóságú, helyi hálózatokra tervezték (akkoriban az Internet és elődei még igen komoly csomagvesztéssel működtek), és mivel nemigen számítottak csomagvesztésre, ezért az UDP protokoll nem nyugtázza a vételt. A feladó elküldi a csomagot, aztán erősen hisz abban, hogy a másik oldal megkapta. Nincs visszajelzés. Az UDP-t ma már az interneten is használják olyan protokollok hordozójaként (mert ő is csak szállító, mint az IP), ahol nagyon fontos a sebesség, de nem számít, ha egy-egy csomag elvesz. Jó példa erre a VoIP, azaz az internetes telefon, azon része, mely a beszédet szállítja. Pompás hibajavító mechanizmusai miatt egy-egy csomag elvesztését a felhasználók nem is hallják meg, a késleltetése viszont igen kellemetlenné teszi a beszélgetést. A TCP olyan mechanizmusokat tartalmaz, melyek kötelezővé teszik a csomagok ellenőr-

zését és nyugtázását, így a TCP protokollt használó alkalmazások (igen, ő is kizárólag szállít) biztosak lehetnek benne, hogy minden elküldött csomag célhoz ér. Ennek érdekében a TCP szükség esetén újraküld csomagokat, ha a túloldal nem igazolja vissza őket adott időn belül. Ezzel párhuzamosan szükség esetén lassítja a csomagok feladási sebességét, hogy ne árássza el a lassabb vevőt. Ez az a tulajdonság, amit a továbbiak megértéséhez meg kellett ismerni. Most szépen elköszönünk a kényelmesebbektől, és leszállunk a protokollok bugyraiba.

3.1. Az ICMP és UDP jellemzői

A számítógépek kommunikációja nagyon hasonlít az emberi beszédre. A nyelv, amit beszélnek, a protokoll. Sokféle felhasználásra az idők folyamán nagyon sok különböző protokoll alakult ki, minket most leginkább az interneten leginkább használt IP, ICMP, UDP, és különösen a TCP érdekel. Az IP-ről fent már leírtuk, amit most tudni érdemes, térjünk rá az ICMP-re. Az ICMP (Internet Control Message Protocol), ahogy a neve is mutatja, az internetes forgalom irányításában segít. Sok feladata van, minket csak egyetlen része érdekel: a Source Quench üzenet. Ennek feladata, hogy közölje az adóval: a vevő nem képes olyan gyorsan venni az adatokat, ahogy ő küldi. Nézzünk egy egyszerű példát: van egy nagyon nagyon gyors szerver, akitől egy gyenge kliens elkezd letölteni valami nagyot. A szerver elkezdi szórni az adatokat, de szegény gyenge kliens nem tudja olyan gyorsan eltárolni az adatokat, ahogy kapja őket. Ilyen esetben küld vissza a kliens IP stack-je egy Source Quench ICMP üzenetet. Erre az adó szépen lecsökkenti a cwnd (lásd TCP protokoll leírás) értékét egy szegmensnyire, és újra kezdődik a slow start. Erről bővebben a TCP protokoll tárgyalásánál. Korábban a kapcsolat útjában lévő router-ek is küldhettek ilyen csomagot, de az aktuális RFC-k szerint már csak a cél küldhet. Az ECN (Explicit Congestion Notification) megjelenése óta a Source Quench jelentősége lassan visszaszorul.

Most ejtsünk szót kicsit bővebben az UDP (User Datagram Protocol) protokollról. Az UDP kapcsolatmentes protokoll, nincs kapcsolatfelépítés és -bontás. Az elküldött csomagok mindenképpen átmennek a csatornán, így a masszív UDP forgalom vevő oldalon nem szabályozható vissza, erre egész egyszerűen nincs semmilyen módszere a vevőnek. Mindez addig teljesen rendben is volt, amíg az interneten csak segédprotokollnak használták az UDP-t. Ilyen felhasználás a domain vagy az ntp. Ma azonban egyre nagyobb teret hódít a VoIP, ahol a csatorna felépítése és a kapcsolat jellemzőinek egyeztetése TCP-n zajlik, de a hang és kép átvitele UDP-n, ezzel komoly fejtörést okozva a sávszélességet szabályozni vágyóknak. Kijelenthető: ha UDP protokollon nagy mennyiségű adattal eltömik a bejövő csatornákat, akkor szépen elmehetünk teázni, mert annak csak akkor szakad vége, ha a küldő akarja. Van azért egy megoldás, ha nem is túl egyszerű, de erről majd később. Most menjünk rá a nagy vadra: a TCP-re.

3.2. A TCP jellemzői

Ma a TCP (Transmission Control Protocol) végzi az internet adattovábbításának az oroslán részét – és szerencsére nagyon okos. A TCP kapcsolat alapú protokoll, ügyel a csomagvesztés elkerülésére, a helyes sorrendre és az ismétlődés mentességre. A protokoll működési mechanizmusai igen összetettek, vaskos kötetek szólnak róla, a sávszélesség szabályzásának szempontjából minket azonban csak néhány dolog érdekel. A TCP egyik legfontosabb fogalma az ablak, mely kettős célt szolgál. Az egyik feladata

az újraadás szabályozása, ez minket most kevésbé érdekel, a másik viszont némi forgalom szabályzás: az ablak a csatornára egyszerre kiküldhető még nem nyugtázott csomagok méretét adja meg. Ha a küldő minden csomagra egyenként megvárna a nyugtát, az adatátvitel nagyon lassú lenne. Az adó ablakában már elküldött, de még nyugtára váró, valamint még el nem küldött csomagok vannak. Amint a vevő nyugtáz egy csomagot, azt az adó kiveszi az ablakból, ahová ezután újabb küldendő adatok kerülnek. Ha a vevő nem növeli megfelelő sebességgel az ablakot, akkor csökkentheti méretét. Ezzel a kliens szabályozhatja az adási sebességet, hiszen ha az ablak méretét nullára csökkenti, akkor az adó csak akkor adhat újabb csomagot, ha nyugtát kapott az előzőről. Így tehát megvalósítható a nagy áram: a bejövő kapcsolatok sávszélessége szabályozható. Ezzel csak annyi gond van, hogy a hálózaton más rendszerek is vannak, így ez néha nem elegendő.

A kommunikációban nagyon fontos timeout-ok megadásakor fontos szerepe van az ún. RTT-nek (round trip time), azaz a vonal késleltetésének. A TCP stack-ek ezt folyamatosan mérik a kapcsolat alatt. Amennyiben pl. egy nyugtát késleltetünk, úgy befolyásoljuk az RTT-t is.

A TCP protokoll egy hasznos eszközt ad a kommunikáció hálózati terheléshez igazításához: ez a slow start algoritmus. Az algoritmus lényege, hogy a normál ablak mellett bevezeti a torlódási ablak (congestion window) fogalmát. Az adási jog a normál és a torlódási ablak méretének minimumától függ. A kapcsolat felépítése után a cwnd értéke egy lesz, amit minden csomag veszteségmentes átvitele után megdupláz az IP stack. Az exponenciális növekedés hatására a kapcsolat gyorsan eléri a maximális sebességet, így a slow start hatására a két rendszer kommunikációja lassan indul, és a vevő, illetve a csatorna lehetőségei szerinti szinten megállítja az adási sebességet. Ez azonban még mindig kevés a torlódás elkerüléséhez, ezért a slow start algoritmust kiegészíti a torlódás elkerülési (congestion avoidance) algoritmus. Az algoritmus lényege, hogy a mai nagy megbízhatóságú hálózatokon a csomagok elvesztése egészen biztosan a torlódás jele lesz, így az algoritmus bevezet egy ssthresh nevű változót, amely két lehetséges működési mód közti váltásra szolgál. Amennyiben az ssthresh kisebb, mint a cwnd, akkor slow start fog indulni, ellenkező esetben a nyugták hatására a cwnd-t csak $1/cwnd$ értékkel fogja növelni. Amikor újraadásra kerül sor, akkor az ssthresh változót beállítja az aktuálisan használt ablak (a cwnd és a normál ablak minimuma) felére. Ha timeout történt, akkor a cwnd-t visszaállítja egyre, és újra kezdődik a slow start. Eddig a csomag eldobásával és az ablak méretének csökkentésével jelezhetette a vevő az adónak, hogy lassítson. Újabban van azonban erre jobb mód is: az explicit congestion notification.

A tcp újraadás bemutatásakor láthattuk, hogy miként kezelhető a torlódás. Azonban több protokoll is létezik, ahol a csomagvesztés miatt megnövekedő késleltetés nagyon zavaró lehet. Ezt a hátrányt könnyen meg lehetne szüntetni, ha a router-ek túl hosszú adási sor esetén nem egyszerűen eldobnák a csomagot, hanem torlódási jelzést tudnának küldeni. E célra dolgozták ki az ECN-t, melynek megvalósítása nem csak a TCP protokollhoz kötött, bármilyen protokoll kiegészíthető vele. A protokoll egy része az IP, míg másik része a TCP rétegben helyezkedik el. Az IP rétegben a kommunikáció az IP fejléc két bitjén át zajlik, erre a diffserv mező melletti két, eddig nem használt bitjét jelölték ki. Az előző ECN leírás itt két külön bitről szólt, a végleges szabvány azonban a két bitet egyben kezeli, az így kapott értéket ecn codepoint-nak hívják. A 00 érték azt mutatja, hogy a csomag feladója nem ismeri az ECN-t, vagy nem akarja azt használni. A 01 az ECT(1), míg az 10 az ECT(0) codepoint. Az 11 codepoint jelentése CE (Congestion Experienced). Amennyiben egy router olyan csomagot vesz, amely az

ECT(0) vagy ECT(1) codepoint-ot tartalmazza és torlódást érzékel, úgy átállítja a CE codepoint-ra.

Azonban így csak a túloldalt sikerült értesíteni, ezt az infót még vissza kell juttatni megbízhatóan a feladónak. Mindez azért fontos, hogy az újraadási policy-t ennek megfelelően szabályozhassák. A szabályzás elősegítésére a TCP fejlécbe bevezettek két új flag-et: ECE (ECN Echo), valamint a CWR (Congestion Window Reduced). Amikor valamely oldal olyan TCP csomagot vesz, amely CE codepoint-ot tartalmaz az IP fejlécbe, úgy a nyugtában beállítja az ECE TCP flag-et. A túloldal ezt olyan módon nyugtázza, hogy a következő csomagjában a CWR TCP flag szerepel. Az ECN jelzések csak olyan kapcsolatoknál használhatóak, ahol már az elején megállapodtak az ECN használatáról. Ez úgy történik, hogy a kapcsolatot felépítő oldal a SYN-es csomagjában az ECE és a CWR bitet is beállítja (*ECN-setup SYN packet*). Amennyiben a túloldal szintén ismeri és használni szeretné az ECN-t, úgy a ECE bitet magasra, a CWR bitet pedig alacsonyra állítja (ezt hívjuk *ECN-setup SYN-ACK packet*-nek).

3.3. A legfontosabb hálózati protokollok sávszélesség jellemzői

HTTP, HTTPS Az interneten leggyakrabban használt protokollok, a webszerverek és kliensek közti kommunikációra és bizonyos esetekben más forgalmakra is (pl. bizonyos esetekben a Skype is használja). TCP szállítja, így kifelé és befelé is jól kontrollálható. Jellemzően kis kifelé és a felhasználástól függően akár hatalmas befelé irányuló forgalmat okoz. Ha web alapú levelező rendszereken csatolt állományokat töltünk fel, vagy valamilyen jelentősebb méretű form-on megnyomjuk a megfelelő gombot, akkor a kifelé irányuló adatmennyiség is jelentős lehet. Általában elegendő a bejövő irányt szabályozni.

FTP Adatátvitelre használt TCP alapú protokoll. Általában letöltésre használják, de képes feltöltésre is. Felhasználástól függően mind a két irányt kontrollálni kell. A TC szempontjából igen kellemetlen tulajdonsága, hogy több TCP kapcsolatot használ, így szűrése csak az Netfilter ftp conntrack moduljával oldható meg kényelmesen. Ennek használatáról majd később szólunk.

SMTP Levél küldésre használt, TCP feletti protokoll. Munkaállomáson szinte kizárólag kifelé irányuló forgalmat okoz, levelező szervereken általában befelé is ezt használják. Felhasználástól függően kell szabályozni, az otthoni gépeken csak kifelé.

POP3, POP3S Levelek letöltésére használják. TCP alapú. Minimális a felfelé forgalma, a lefelé viszont a levelek mennyiségétől függően akár nagyon nagy lehet. A bejövő forgalom szabályozása mindenképpen ajánlott.

IMAP, IMAPS Postaládák elérésére használható, TCP-n közlekedik. Alap esetben a leveleknek csak a fejlécei jönnek át a csatornán, és a levél törzse csak az elolvasás előtt. Általában ritkán mentünk a helyi leveles ládából az IMAP szerverre, de ilyenkor jelentős felfelé menő forgalmat is okoz. Általában a bejövő forgalmának szabályozása szükséges.

SSH Távoli hozzáférésre használt, TCP szállítja. Alapvetően parancsok kiadására használható, de lehet rajta keresztül fájlokat másolni (scp, sftp) TCP portokat továbbítani (forwarding) és SOCKS proxyként is használható (-D opció). Így tehát a kifelé és befelé irányuló forgalma is a használat módjától függ, így általában mindkét irányban érdemes szabályozni.

VoIP protokollok Napjaink egyre népszerűbb lehetősége az internet telefon. Az interneten hang és képátvitelre használt protokollok (pl. H.323, SIP stb.) több csatornát használnak, TCP-t és UDP-t egyaránt. Forgalmuk szabályozása nehézkes, bizonyos esetekben (bejövő szabályzás) lehetetlen. A bejövő forgalom szabályzására az egyetlen használható módszer egy külső router használata az interneten, ahol a befelé irányuló VoIP UDP forgalmat kifelé menőként, queue-kkal lehet szabályozni.

egyéb, kicsi de fontos forgalmak Sáv szélesség szempontjából nem jelentős, de kellemetlen problémákat okozhat a domain és az ntp forgalom fennakadása. Mindkettő jellemzően UDP-t használ (a domain esetén zónatranszferre használnak TCP-t, de ezt a névszerver üzemeltetők nyilván tudják). Ezeknek a forgalmaknak érdemes egy kicsi de fix csatornát biztosítani a fennakadás mentes működés miatt.

egyéb Ha a hálózaton egyéb forgalmak vannak, akkor azokat egyenként meg kell vizsgálni. Erre egyszerűen használható például az ntop program.

4. A sáv szélesség-menedzsmentről

A hálózati kapcsolat két „csőként” is elképzelhető, az egyik a kimenő forgalmat viszi, a másik a bejövőt. Világos, hogy a kimenő csőbe olyan sorrendben és olyan logika szerint küldjük az adatokat, ahogy akarjuk. Ha a kimenő sáv szélesség szabályzását szeretnénk megoldani, akkor erre minden lehetőségünk megvan (a Linux biztosítja ezeket). A bejövő csőnél viszont könnyen belátható, hogy közvetlenül csak akkor befolyásolhatjuk a bejövő adatok protokollok és gépek közti elosztását, ha a cső másik végén állítani tudjuk a csőbe továbbított adatokat. Erre általában nincs lehetőség, az internet szolgáltatók csak igen különleges esetekben működnek közre ilyen megoldás felállításában. Ez tehát kiesik. Szerencsére azonban az internet forgalmának jelentős része TCP protokollon zajlik, és – ahogy az előző fejezetben láthattuk – ennek a bejövő sebességét indirekt módon befolyásolhatjuk.

4.1. A forgalom szabályzásának módszerei

shaping A csomagok várakoztatása az elvárt átlagos sebesség eléréséig. Kizárólag kimenő forgalomra használható, mivel a bejövő csomagokat nem tudjuk a csatornán várakoztatni, azok bizony bejönnek.

scheduling Az időzítés (vagy más néven újrendezés (reordering)) lehetővé teszi a csomagok átadási sorrendjének megváltoztatását.

policing Valamilyen akció végrehajtása minden csomagon, amely túllépi az elvárt sáv szélességet. Általában ez a csomag eldobását jelenti, ami TCP esetén meglepően hatásos módszer, mint azt korábban láttuk. UDP esetén általában semmi másra nem jó, csak csomagvesztés előidézésére.

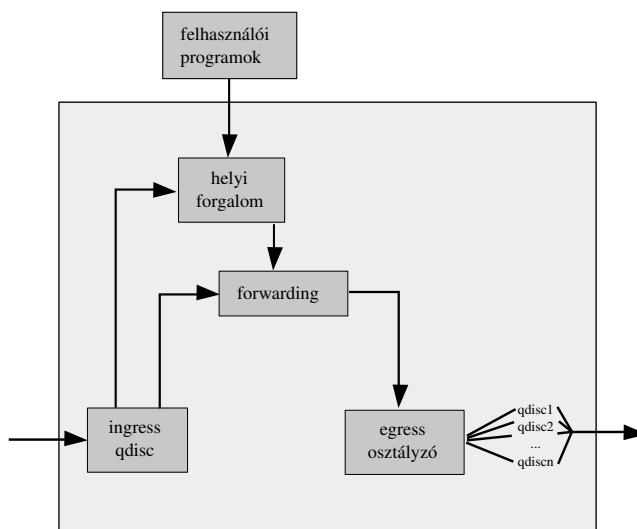
A kimenő forgalom szabályzására használható a shaping, ahol a kimenő csomagokat a hálózati TC alrendszer valamilyen szabályrendszer alapján várakoztatja a szükséges ideig. Ehhez sorokat használ, ahová a beérkező csomagokat besorolja. A bejövő forgalom jelenleg csak policing módszerrel szabályozható, ahol jelenleg nincs lehetőség

osztály alapú sorba állítási módszerek használatára, bár némi kiegészítéssel vagy trükközéssel a bejövő forgalomra is használhatunk osztály alapú sorba állítást, de erről majd később.

5. A Linux TC alapjai

Arról már szóltunk, hogy sorok segítségével befolyásolhatjuk a csomagok kiküldését. A kernel a rendelkezésünkre bocsát számos különböző képességekkel rendelkező besorolási algoritmust, az ún. *qdisc*-eket, a kimenő (egress) és a bejövő (ingress) forgalom szabályozásához. Ezek segítségével tudjuk módosítani az adatok továbbításának módját. Két csoportba sorolhatók. Vannak osztály alapú (classful – CF) és osztálymentes (classless – CL) besoroló módszerek. Az osztálymentes módszerek képesek a csomagok fogadására, újraütemezésére, várakoztatására vagy eldobására. Az osztály alapú besoroló módszerek több osztályt tartalmazhatnak, melyek újabb besorolási módszereket tartalmazhatnak (CL vagy CF) és így tovább. Az osztályok származhatnak (parent) egy *qdisc*-ből vagy más osztályokból. Levél osztály alatt olyan osztályokat értünk, melyeknek nincs alosztálya (child), és rendelkezik egy *qdisc*-kel. Mikor létrehozunk egy osztályt, akkor alpból egy fifo *qdisc* kapcsolódik hozzá (lásd később), mely gyerekosztályok hozzákapcsolásánál automatikusan megszűnik. A levélosztályok *qdisc*-jeit bármikor le lehet cserélni CL vagy CF *qdisc*-re.

Az osztályoknak szükségük van egy algoritmusra, amely megadja, hogy hogyan kell a beérkező csomagokat elosztani az alosztályok között, ez az osztályzó (classifier). Az osztályzás szűrők (filter) alapján végezhető el. Egy szűrő valamilyen feltételek szerint eldönti, hogy egy adott csomag hozzá tartozik-e vagy nem. Az 1. ábra a hálózati forgalom logikai útját mutatja a Linuxban.

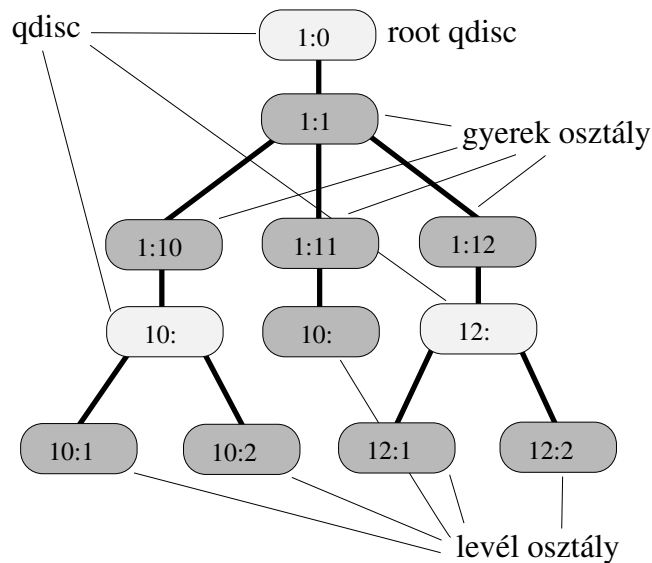


1. ábra. A forgalom útja a TC rendszerben

A bal oldalon látható a hálózatról beérkező forgalom, amely egy speciális *qdisc*-be érkezik, ez az *ingress qdisc*. Itt végezhető el a beérkező csomagok kezelése. Ehhez szűrők köthetők, melyek lehetőséget adnak a bejövő csomagok eldobására, ezzel a TCP

forgalom lassítására. Ha a csomag nem dobódott el, akkor ezek után következik a routing döntés, ahol a kernel eldönti, hogy a csomag helyi folyamathoz tartozik (ekkor az megkapja), vagy tovább kell küldeni egy másik csatolón. Utóbbi esetben a kimenő hálózati interfészhez kapcsolt qdisc folytatja a csomag feldolgozását és a beállításainak megfelelően kezeli azt.

Minden csatolóhoz tartozik egy kimenő (egress) ún. root qdisc. Alapesetben ebbe „esnek be” a csomagok, ha csak valamelyik osztályzó nem sorolja őket máshová. Minden qdisc-nek van egy azonosítója (handle), amely két számból, a *major*-ból és a *minor*-ból áll. A major-t a felhasználó adja meg, a qdisc-ek esetén minor minden esetben 0. Egy root qdisc esetén az azonosító valahogy így néz ki: *1:0*. Az osztályok major-jainak meg kell egyeznie a szülőosztályok major-jával. A major-nak egyedinek kell lennie az ingress-en és az egress-en is. Az osztályok logikai származási fájára mutat egy példát a 2. ábra.



2. ábra. Egress struktúrafa

A fa nem a besorolást segíti elő, hanem, ahogy később látni fogjuk, a segítségével a csatorna egyszerűen és logikusan felosztható a különböző osztályok között. Ezt úgy oldják meg, hogy minden osztály csak a felette állóval beszélhet, csak az ő sávszélességéből gazdálkodhat. Az osztályba sorolást a szűrők végzik el, minden osztályhoz kapcsolhatóak szűrők, és segítségükkel minden csomag átdobható egy másik osztályba.

5.1. A Linux TC kernel részei

Ennyi bevezető után ideje lenne rátérni a lényegre. Ha befolyásolni szeretnénk a sávszélességet, akkor szükségünk lesz a kernel támogatására. Az alábbiakat kell a kernelbe fordítanunk (csak a most használt részeket említve):

```

Networking ->
[*] Networking support
Networking options --->

```

```

<M> Firewall based classifier
<M> U32 classifier
[*] QoS and/or fair queueing ---->
    <M> HTB packet scheduler
    <M> The simplest PRIO pseudoscheduler
    <M> SFQ queue
    <M> TBF queue
    <M> Ingress Qdisc

```

A fenti alrendszerket a komolyabb disztribúciók befordítják a rendszermagba, így ezzel általában nem kell bajlódniuk. A Linux TC beállításához szükségünk lesz még a tc parancsra is, amely általában az iproute2 csomag része.

5.2. Besorolási módszerek (qdisc)

Az írás mérete nem teszi lehetővé, hogy minden létező módszert bemutassunk, így egyszerűen csak egy könnyen használható, világos megoldás kivitelezéséhez szükséges részeket ismertetjük. Amennyiben valakinek ettől bonyolultabb igényei vannak, akkor úgysem ússza meg a LARTC elolvasását.

5.3. Osztálymentes qdisc-ek (classless)

pfifo_fast

A legegyszerűbb, minden hálózati csatolóhoz alapértelmezetten kapcsolt qdisc. A forgalmat három sávra (band) osztja, minden sáv egy FIFO, ami azt jelenti, hogy az elsőként betett csomag hagyja el először a sávot. A forgalmak sorokba osztását az IP csomag TOS és PRECEDENCE mezői alapján oldja meg, például a Minimum Delay jelzővel ellátott csomag a 0. sorba kerül, és így tovább. A javasolt TOS besorolásokat az RFC-1349 írja le. A sávok egymáshoz képesti kezelése úgy zajlik, hogy amíg a 0. sávon van csomag, addig az 1. sávot nem hagyja el csomag, és így jár el az 1. és 2. viszonylatában is. Ebből azonnal látszik egy kellemetlen mellékhatása: ha egy nagy sávszélességet igénylő forgalom csomagjai rendre Minimum Delay jelzővel vannak ellátva, akkor az lehetetlenné teszi a többi forgalom áthaladását. Ezt a hatást remekül meg lehet figyelni az ssh port továbbítása esetén, ha az átvitt csatornában nagy mennyiségű adatot viszünk át (például egy http port továbbítása esetén). Ilyenkor az ssh képes minden más, jelzővel nem ellátott forgalom elnyomására.

TBF – Token Bucket Filter

A TBF olyan egyszerű, jól használható qdisc, amely lehetővé teszi a rajta átmenő forgalom lassítását. Nagyon precíz, nem igényel sok processzoridőt, valamint hálózatbarát. A hálózati forgalom lassítására kiválóan használható. Csak akkor adja tovább a csomagokat, ha azok továbbításával nem haladja meg az engedélyezett sávszélességet. Az implementáció tartalmaz egy puffert (bucket), amely megadott sebességgel, folyamatosan töltődik virtuális adatdarabkával, ún. tokenekkel. A TBF legfontosabb paraméterei a puffer mérete, amely megadja, hogy hány token-t lehet tárolni benne, illetve a korábban említett töltési sebessége. Minden bejövő adatcsomagot egy token-hez kapcsol, amit eltávolít a pufferből. Amennyiben az adatok éppen a megadott sebességgel érkeznek, akkor a pufferben mindig lesz hozzájuk kapcsolható token, ha lassabban,

akkor a puffer tele lesz tokenekkel, tehát az adatsomagok azonnal továbbítódnak. Látható, hogy ha az adatok lassabban érkeznek, akkor kisebb lökések (burst) akár meghaladják a beállított sebességet, ha azonban az adatok gyorsabban érkeznek, mint a megadott sebesség, akkor a pufferből hamarosan kifognak a token-ek, így a beérkező csomagok továbbítása átmenetileg szünetel, tehát a sávszélesség a beállított szinten marad. A legfontosabb beállítható jellemzői tehát:

rate A sebességhatár.

burst A puffer mérete bájtokban megadva. Vigyázat! Ez megadja egy lehetséges adatlökés maximális méretét. Ha nagyon kicsi, akkor lehetetlenné teszi az adatok továbbítását. A gyakorlat azt mutatja, hogy ha nagyságrendileg nem hasonló a rate értékéhez, akkor a TBF nem tudja a beállított sebességet átvinni. Ennek oka a hálózati forgalom sajátosságaiban gyökerezik, ugyanis az soha nem egyenletes. Ha a puffer méretét a rate méretére (vagy nagyobbra) állítjuk, akkor a maximális átlagos sebesség biztosan a beállított lesz.

limit A csomagok sorának mérete bájtokban. Nem kötelező meghatározni.

latency Megadja, hogy egy csomag legfeljebb mennyi ideig lehet a sorban. Nem kötelező meghatározni.

SFQ – Stochastic Fairness Queueing

Az SFQ olyan qdisc, amely igyekszik igazságosan elosztani a hálózatot a beérkező kapcsolatok között. A TCP kapcsolatok és UDP adatfolyamok alapján a beérkező csomagokat nagyszámú sorba osztja szét, és utána minden sorból legfeljebb a quantum által meghatározott mennyiséget továbbítja. Ezzel elérhető, hogy a gyorsabb kapcsolatok nem tudják elnyomni a lassabbakat. Természetesen csak akkor van értelme, ha a hálózati csatoló teljesen ki van használva, mert ha nincs, akkor az SFQ nem csinál semmit, hiszen minden kimenő csomagot azonnal továbbítani lehet. No de kinek nincs állandóan kitömve a kimenő vonala? Az SFQ legfontosabb paraméterei:

perturb Az elosztási besorolás újrakonfigurálásának ideje. Amennyiben nincs megadva, akkor soha nem lesz újrakonfigurálva, ami nem jó ötlet. A 10 másodperc jó választás.

quantum A sorból egyszerre kiküldhető adat mennyisége bájtokban. Az alapbeállítása egy csomag mérete. Ne állítsuk az MTU mérete fölé!

limit A sorba állítható csomagok maximális száma. Amennyiben eléri, akkor elkezd eldobni a csomagokat.

5.4. Osztály alapú qdisc-ek (classful)

Itt csak egyetlen, de nagyszerűen használható qdisc-ről szólunk. A neve HTB (Hierarchical Token Bucket), és a gyakran használt, de kissé bonyolult CBQ helyettesítésére hozták létre. A HTB tulajdonképpen a TBF osztály alapú változata. Legfontosabb beállítási jellemzői:

rate Meghatározza a token-ek előállítás sebességét és így az átlagos sávszélesség határt is (lásd még TBF). A root osztálynak a vonal sávszélességét kell megadni.

burst A puffer mérete (lásd még TBF).

ceil Az osztály által használható legnagyobb sávszélesség.

prio Az alacsony várakozási időt igénylő osztályok prioritizálására használható, például hang és kép átvitele esetén, vagy távoli shell használatánál. Paramétere egy szám. A magasabb besorolású osztály kap először a sávszélességből (természetesen csak a rate és a ceil által meghatározott mértékig).

default Az alapértelmezett gyerekosztály azonosító minorja. Hacsak egy szűrő másként nem dönt, akkor ide kerül a csomag.

A root osztályon kívül minden HTB képes sávszélességet kölcsönvenni a testvéreitől. Ám ha közvetlenül a root alá vesszük fel a gyerekosztályokat, akkor azok sem tudnak kölcsönözni, így általában érdemes a root alá felvenni egy osztályt, és a többieket ez alá tenni. Így szükség és lehetőség esetén a levélosztályok már kaphatnak sávszélességet a többiektől.

Az osztályhierarchia megfelelő felépítésével, a ceil és a rate megfelelő megválasztásával szinte minden, gyakorlatban felmerülő feladat megoldható. Ezért használjuk ezt a qdisc-et a példáinkban.

5.5. A forgalom osztályozása

A forgalom osztályozására két leggyakrabban használt szűrő a *firewall* és az *u32*. A *firewall* szűrő a csomagszűrő által beállított jelek (mark) alapján osztályozza a csomagokat, ilyen jel helyezhető fel a mangle táblában a *-j MARK -set-mark <jel>* akció segítségével. A Netfilter hatalmas tudása és rugalmassága ezáltal felhasználható a TC rendszer kialakításánál is. A legjobb az OUTPUT tábla elejére betenni egy jelölő chaint, ami a szükségleteknek megfelelően megjelöli a csomagokat. Érdemes megjegyezni, hogy a bejövő hálózati eszközön is megjelölhetjük a csomagokat, és ez alapján osztályozhatjuk a kimenő oldalon. Ennek felhasználása a szűrőben:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 handle 10 \
  fw flowid 1:1
```

Ez a *10* jelzéssel ellátott csomagokat az *1:1* azonosítójú osztályhoz rendeli. Az *u32* szűrő segítségével közvetlenül a csomag jellemzőire szűrhetünk, és ezek alapján irányíthatjuk a csomagokat. Az *u32* legfontosabb paramétere:

src Forráscím vagy hálózat. Használata: *'match ip dst 10.1.2.0/24'*, amely a 10.1.2.0 hálózati című, C osztályú hálózatra illeszkedik. Ha egy bizonyos gépet szeretnénk megadni, akkor a maszk legyen 32.

dst Célcím vagy hálózat. Használatát lásd a forráscímnél.

sport Forrásport. Használata: *'match ip dport 110 0xffff'*, amely a 110-es portról jövő (azaz POP3) csomagokra illeszkedik.

dport Célport. Használatát lásd a forrásportnál.

protokoll Protokollra a */etc/protocols* fájlban megadott szám segítségével egyszerűen szűrhetünk így: *'match ip protocol 1 0xff'*. Ez az ICMP, vagyis az 1 azonosító számú csomagokra illeszkedik.

Tehát egy összetett u32 szűrő valahogy így néz ki:

```
# tc filter add dev eth0 parent 1:0 protocol ip prio 1 u32 match ip src \
  192.168.1.1 match ip dport 25 0xffff flowid 1:1
```

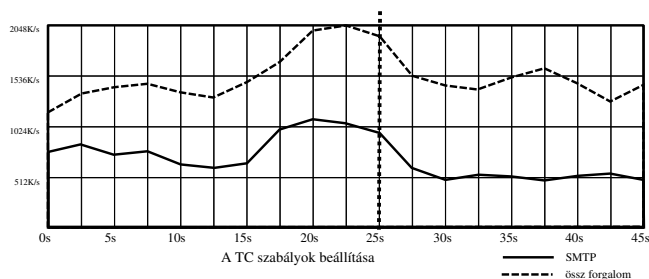
amely a 192.168.1.1 címre menő, 25-ös portra irányuló forgalmat bedobja az 1:1 azonosítójú osztályba.

6. A kimenő forgalom szabályozása

Most jöjjön a gyakorlat. A kimenő forgalom szűrését a következő módon állítjuk be:

```
# tc qdisc add dev eth0 root handle 1: htb default 10
# tc class add dev eth0 parent 1: classid 1:1 htb rate 2mbit burst 10k
# tc class add dev eth0 parent 1:1 classid 1:10 htb rate 1536kbit \
  ceil 2mbit burst 10k prio 1
# tc class add dev eth0 parent 1:1 classid 1:20 htb rate 512kbit \
  ceil 512kbit burst 10k prio 2
# tc qdisc add dev eth0 parent 1:10 handle 10: sfq perturb 10
# tc qdisc add dev eth0 parent 1:20 handle 20: sfq perturb 10
# tc filter add dev eth0 parent 1: protocol ip prio 1 handle 120 fw \
  flowid 1:20
# iptables -t mangle -I OUTPUT -o eth0 -p tcp --sport 25 -j MARK \
  --set-mark 120
```

Ezek mellett a beállítások mellett a rendszer sávszélességet ketté fogja vágni az 1:10 és az 1:20 azonosítójú HTB osztályra. Az 1:10 osztály maximális sávszélessége 1536kbit/s, és ha a vele egy szinten lévő osztály tud adni, akkor 2Mbit/s-ig kölcsön kéri annak sávszélességét. Ebbe fog bekerülni minden olyan forgalom, ami nem lesz explicit módon osztályba sorolva. A másik osztály, amelynek azonosítója 1:20 sávszélessége 512kbit/s, és nem kér kölcsön a többiekől. Az osztályozást firewall szűrővel végezzük, a Netfilter jelek alapján. Látható, hogy a kimenő forgalomnak a 25-ös portra menő része (a kimenő SMTP forgalom) lesz 120-as jellel ellátva, amit a firewall szűrő „bedob” az 1:20 osztályba. A 3. ábra mutatja a hálózat kihasználtságát a beállítás előtt és után. Mindkét levélosztály egy-egy SFQ qdisc-et kapott. A végeredmény: a kimenő levelezés nem tudja elfoglalni a teljes sávszélességet, csak annak egy negyedét.



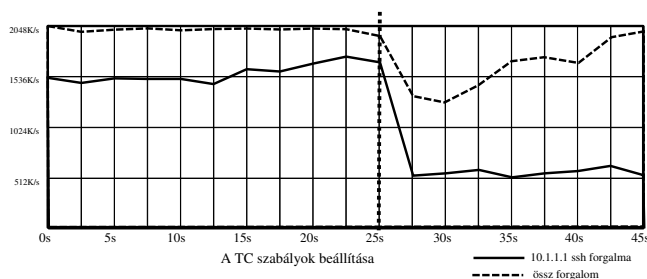
3. ábra. A kimenő forgalom szabályozása

7. A bejövő forgalom szabályozása

A betöltött szabályrendszer nagyon egyszerű:

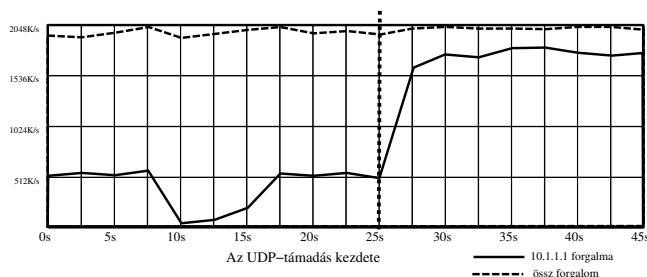
```
# tc qdisc add dev eth0 ingress
# tc filter add dev eth0 parent ffff: protocol ip prio 50 \
  u32 match ip dst 10.1.1.1/32 police rate 512kbit burst \
  512kbit drop flowid :1
```

Ez létrehoz egy ingress root qdisc-et az eth0 csatolóra, amely valójában nem klasszikus qdisc, csak annyira képes, hogy szabályok alapján eldobjon csomagokat. Egyetlen szabály lesz, ez a 10.1.1.1 gépről jövő forgalmat csökkenti 512kbit/s-ra. Minden olyan csomag, amely ezt a sebességet meghaladná, el lesz dobva. A többi bejövő forgalom nem lesz kezelve, így azok tetszés szerint kitölthetik a csatornát. Ahogy a 4. ábrán látjuk, ez SSH esetén (és nyilván a többi TCP alapú protokoll esetén is így lenne) szépen működik is.



4. ábra. A bejövő forgalom szabályozása

Ha azonban a 10.1.1.1 irányából nagy mennyiségű UDP forgalom érkezik (az nc segítségével állítottuk elő), akkor a szabályzás teljesen tönkremegy, ahogy azt az 5. ábrán láthatjuk.



5. ábra. A lökésszerű UDP-forgalom hatása bejövő TC esetén

Ahogy már korábban is említettük, a bejövő forgalomra nem lehet várakozási sorokat használni, mivel a csatornán nem várhatnak a csomagok. Ezt a hiányosságot többféle módon megkerülhetjük. Az egyik lehetőség a virtuális gépek használata (pl. User Mode Linux), így a bejövő forgalom a gépen belül egy virtuális hálózati csatolón (pl

tun eszközön) távozik, és itt már tehetünk hozzá várakozási sorokat tartalmazó qdisc-eket is. A másik módszer jelenleg nem része a hivatalos kernelnek, ez az IMQ. Ez egy virtuális hálózati eszköz, amely használatával a bejövő forgalom kimenőként láttatható és így kezelhető.

Figyelmeztetés: A bejövő forgalom szabályozása csak akkor oldható meg tökéletesen, ha az interneten elhelyezünk egy kis Linux gépet, aki fogadja a befelé irányuló forgalmat (valójában mindenki azt hiszi, hogy az ő IP címe a miénk), és a kimenő csatolóján végzi el a forgalom szabályzását.

Az előadáson bőszegesebb gyakorlati példákat mutatunk az itt leírtak használatára egyszerűbb és összetettebb szituációkban. Ennek a cikknek a kibővített, elektronikus változata elérhető lesz az előadás után a <http://www.andrews.hu/> oldalon.

Hivatkozások

- [1] Wikipedia: <http://hu.wikipedia.org/>
 - [2] Linux Advanced Routing and Traffic Control Howto:
<http://lartc.org/howto/>
 - [3] Differentiated Service on Linux HOWTO:
<http://www.opalsoft.net/qos/DS.htm>
 - [4] Iproute2: <http://linux-net.osdl.org/index.php/Iproute2/>
 - [5] RFC-1349: <http://www.faqs.org/rfcs/rfc1349.html>
 - [6] Nmap: <http://www.insecure.org/nmap/>
 - [7] HTB: <http://luxik.cdi.cz/devik/qos/htb/>
 - [8] IMQ: <http://www.linuximq.net/>
 - [9] User Mode Linux: <http://user-mode-linux.sourceforge.net/>
-