ADSL sávszélesség-gazdálkodás HOGYAN

Dan Singletary

dvsing@sonicspike.net

A dokumentum leírja, hogyan állítsunk be egy Linux útválasztót, hogy hathatósabban kezelje a kimenő forgalmat egy ADSL modemen vagy más olyan eszközön, ami hasonló sávszélesség-tulajdonságokkal rendelkezik (kábelmodem, ISDN stb.). Hangsúlyt fektettünk az interaktív forgalom várakozási, lappangási idejének csökkentésére még akkor is, ha a feltöltési és/vagy letöltési sávszélesség teljesen telített.

Tartalomjegyzék

1. Bevezetés 2

1.1. A dokumentum új verziói 2
1.2. Levelezőlista 2
1.3. A felelősség teljes elhárítása 2
1.4. Szerzői jog és licenc 2
1.5. Visszajelzések és javítások 2
1.6. Magyar fordítás 2

2. Háttér 2

2.1. Előzetesen szükséges dolgok 3
2.2. Elrendezés 3
2.3. Csomagok várakozósorai 4

3. Hogyan működik? 5

3.1. A HTB alkalmazása a kimenő forgalom visszafogására 5
3.2. Prioritásos várakozósor kialakítása HTB-vel 5
3.3. A kimenő csomagok osztályozása iptables-szel 6
3.4. Még néhány fogás... 7
3.5. Próbálkozás a bejövő forgalom visszafogására 7

4. Megvalósítás 8

4.1. Kikötések 8
4.2. Szkript: myshaper 8

5. Az új várakozósor tesztelése 14

6. Rendben, működik! Hogyan tovább? 14

7. Kapcsolódó hivatkozások 14

1. Bevezetés

Jelen dokumentum célja, hogy ajánljon egy módszert egy internethez kapcsolódó ADSL (vagy kábel) modemen kimenő forgalom kezeléséhez. A probléma az, hogy a legtöbb ADSL vonalat lekorlátozták 128kbs vagy e körüli adatfeltöltési sebességre. Még súlyosabbá teszi a problémát a csomagok várakozási sora az ADSL modemen belül, ami ha tele van, 2 vagy 3 másodpercet is igénybe vesz, míg kiürül. Ez együttesen azt jelenti, hogy amikor a feltöltési sávszélesség teljesen telítve van, a többi csomagnak 3 másodpercet is igénybe vehet, amíg kijutnak az Internetre. Ez megbéníthatja az interaktív alkalmazásokat, mint a telnet vagy a többszereplős játékok.

1.1. A dokumentum új verziói

Mindig megtalálod a jelen dokumentum legújabb verzióját a világhálón a http://www.tldp.org webhelyen.

Az új verziók ezen kívül különböző Linux WWW és FTP szerverre is fel vannak téve, beleértve az LDP honlapját a http://www.tldp.org webhelyen.

1.2. Levelezőlista

Az ADSL sávszélesség-gazdálkodást illető kérdések és friss információk vonatkozásában kérlek, iratkozz fel a téma levelezési listájára a http://jared.sonicspike.net/mailman/listinfo/adsl-qos honlapon.

1.3. A felelősség teljes elhárítása

Sem a szerző, sem a terjesztők, sem más közreműködő munkatárs nem felelős semmilyen módon a fizikai, pénzügyi, morális vagy bármely más típusú kárért, amit a szövegben ajánlott dolgok követése okozott.

1.4. Szerzői jog és licenc

1.5. Visszajelzések és javítások

Ha kérdéseid vagy ajánlásaid vannak a dokumentumhoz kapcsolódóan, nyugodtan lépj kapcsolatba a szerzővel a dvsing@sonicspike.net (mailto:dvsing@sonicspike.net) e-mail címen.

1.6. Magyar fordítás

A magyar fordítást Szíjjártó László (mailto: laca[AT]janus.gimsz.sulinet.hu_NO_SPAM) készítette (2002.07.28). A lektorálást Daczi László (mailto:dacas[AT]freemail.hu_NO_SPAM) végezte el (2002.09.05). A dokumentum legfrissebb változata megtalálható a Magyar Linux Dokumentációs Projekt (http://tldp.fsf.hu/index.html) honlapján.

2. Háttér

2.1. Előzetesen szükséges dolgok

A dokumentumban vázolt módszernek működnie kell más Linux konfigurációkon belül is, de nem teszteltük máson, csak a következők alatt:

- Red Hat Linux 7.3
- 2.4.18-5 Kernel teljes QoS támogatással (modulok: OK) és beleértve a következő kernel-foltokat (amik történetesen az újabbakban benne is lehetnek már):
  - HTB várakozósor - http://luxik.cdi.cz/~devik/qos/htb/

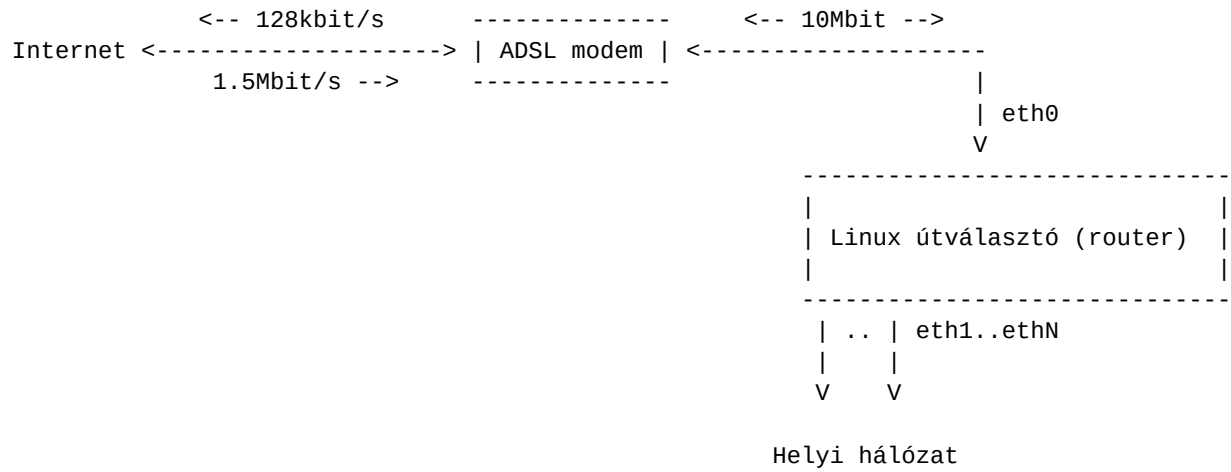
    Megjegyzés: jelentették, hogy a 2.4.18-3 kernelverziót követően a Mandrake (8.1, 8.2) már a HTB-hez adott folttal szállítja a rendszert.

  - IMQ eszköz - http://luxik.cdi.cz/~patrick/imq/
- iptables v1.2.6a vagy későbbi (a Red Hat 7.3-mal szállított iptables verzióból hiányzik a "length" modul)

  Megjegyzés: a dokumentum előző verzióiban olyan sávszélesség-kezelési módszert adtunk meg, ami magában foglalta a meglévő sch_prio várakozósor megfoltozását. Később úgy találtuk, hogy ez a folt teljesen felesleges. Ezen felül a jelen dokumentumban taglalt módszer jobb eredményt ad (bár a doksi írása idején *2* kernelfolt szükséges :) Szerencsés foltozást.)

2.2. Elrendezés

A dolgok egyszerűsítése érdekében, a dokumentumban az összes hálózati eszközre és beállításra vonatkozó hivatkozás a következő hálózati elrendezést tükrözi:



2.3. Csomagok várakozósorai

A csomagok várakozási sorai (queue) olyan "edények", amik az adatokat tárolják a hálózati eszköz számára, amikor azokat nem lehet azonnal elküldeni. A legtöbb várakozósor egy FIFO ("ami elsőnek megy be, elsőnek megy ki") fegyelmezési rendszert, diszciplínát használ (röviden qdisc - a ford.), hacsak speciálisan másra nem állítják be. Ez azt jelenti, hogy amikor egy eszköz várakozósora teljesen tele van, a várakozási sorba utoljára került csomagot csak akkor továbbítja az eszköz, amikor az összes többit már elküldte.

2.3.1. A kimenő ág

ADSL csatlakozás esetén a sávszélesség aszimmetrikus, tipikusan 1.5Mbit/s a bejövő és 128kbit/s a kimenő ág teljesítménye. Bár ez a vonali sebesség, a Linux útválasztó PC és az ADSL modem közötti illesztő tipikusan 10Mbit/s vagy a feletti sebességet tud. Ha a helyi hálózat felé néző csatoló szintén 10Mbit/s sebességű, akkor tipikusan NEM LESZ várakozósor az útválasztónál, amikor a helyi hálózat küld csomagokat az Internet felé. Az eth0 eszközön keresztül olyan sebességgel mennek ki a csomagok, ahogy a helyi hálózatból érkeztek. Ehelyett viszont a csomagok beállnak a sorba az ADSL modemnél, mivel 10Mbit/s-el érkeznek, és csak 128 kbit/s-el mehetnek ki. Időlegesen a csomagok várakozósora a modemnél megtelhet, és minden további csomag, amit küldenek neki, csendben eldobásra kerül. A TCP protokollt úgy tervezték, hogy kezelje ezt, és be fogja állítani a küldési ablak (transmit window) méretét úgy, hogy teljesen kihasználja a rendelkezésre álló sávszélességet.

Amíg a várakozósorok a TCP-vel kombinálva a sávszélesség lehető legjobb kihasználását teszik lehetővé, a nagy FIFO sorok megemelik az interaktív forgalom lappangási idejét.

Egy másik, a FIFO-ra hasonlító várakozósor az n-sávos prioritásos sor. Ennél ahelyett, hogy csak egy várakozósort alakítanánk ki a bejövő csomagok számára, az n-sávos sornak n darab FIFO sora van, amibe a csomagokat az osztályozásuk alapján helyezzük be. Minden sornak van egy prioritása, és a csomagok mindig a legnagyobb prioritású, csomagokat tartalmazó sorból jönnek ki. Ezt a fegyelmezési szabályt alkalmazva az FTP csomagok egy alacsonyabb prioritású sorba helyezhetők, mint a telnet csomagjai, így még egy FTP feltöltés alatt is, egy darab telnet csomag is kijuthat a sorból és azonnal továbbküldésre kerülhet.

A dokumentumot átalakítottuk az új linuxos várakozósor, a Hierarchical Token Bucket (HTB) használatához. A HTB sor nagyban hasonlít a fent leírt n-sávos sorra, de megvan az a tulajdonsága, hogy képes minden osztályának forgalmát korlátozni. Ezen kívül képes arra, hogy forgalmi osztályokat alakítson ki más osztályok alatt, egy hierarchikus osztályokból álló rendszert létrehozva. A HTB teljes leírása meghaladja a dokumentum kereteit, de további információ található a http://www.lartc.org webhelyen.

2.3.2. A bejövő ág

Az ADSL modembe befelé érkező forgalom a kimenőhöz hasonló módon áll várakozósorba, azonban a sor a szolgáltatódnál helyezkedik el. Emiatt valószínűleg nincs közvetlen befolyásod arra, hogyan álljanak sorba a csomagok vagy melyik fajta forgalom kapjon megkülönböztetett kezelést. Az egyetlen mód a várakozási idő alacsonyan tartására itt az, hogy megbizonyosodsz, miszerint nem küldik az adatokat túl gyorsan számodra. Sajnos nincs mód az érkező csomagok sebességének közvetlen befolyásolására, de mivel a forgalmazás többsége valószínűleg TCP, van néhány módja a küldők lelassításának:

- Szándékosan eldobni a bejövő csomagokat - a TCP protokollt úgy tervezték, hogy kihasználja a rendelkezésre álló teljes sávszélességet, miközben próbálja elkerülni a kapcsolaton belül a torlódást. Ez azt jelenti, hogy nagy mennyiségű adat küldésekor a TCP több és több adatot küld, amíg végül is egy csomag eldobásra kerül. A TCP érzékeli ezt, és csökkenti az átviteli ablak méretét. Ez a folyamat ismétlődik a kapcsolat alatt, így biztosítja az adatok lehető leggyorsabb átvitelét.
- A meghirdetett vételi ablak manipulálása - A TCP forgalmazás alatt a fogadó oldal az ACK (elfogadás) csomagok folyamatos sorát küldi vissza. Az ACK csomagokban található az ablakméret meghirdetése, ami kifejezi annak a nem elfogadott adatnak a mennyiségét, amit a fogadó küldeni tud. A kimenő ACK csomagok ablakméretének babrálásával szándékosan lelassíthatjuk a küldőt. Ennek a folyamatszabályozásnak jelen pillanatban nincs (szabadon elérhető) megvalósítása Linuxon (de lehet, hogy dolgozom egyen!)

3. Hogyan működik?

Két alapvető lépéssel optimalizálhatjuk a kifelé menő sávszélességet. Először találnunk kell egy módot arra, hogy megakadályozzuk az ADSL modemet a csomagok sorba állításában, mivel nincs ráhatásunk a várakozósor kezelésére. Ennek érdekében visszafogjuk az útválasztó által az eth0-n kiküldött adat mennyiségét, kicsit kevesebbre, mint a modem teljes kimenő sávszélessége. Ez azt eredményezi, hogy az útválasztó rakja várakozósorba a helyi hálózatról érkező csomagokat, amik gyorsabban érkeznek, mint megengedett kiküldésük.

A második lépés egy prioritásos várakozósor-fegyelmi szabály felállítása az útválasztón. Meg fogunk valósítani egy olyan sort, amit be lehet állítani úgy, hogy elsőbbséget adjon az interaktív forgalomnak, mint a telnet vagy a többszereplős játékok.

> A HTB várakozósor használatával meg tudjuk valósítani a sávszélesség korlátozását és prioritásos várakozósort, egyidejűleg meggyőződünk, hogy nincs olyan osztály, amit egy másik "kiéheztetne". A kiéheztetés elkerülése érdekében nem választható a dokumentum 0.1-es javításánál megadott módszer.

A végső lépés a tűzfal beállítása, hogy az fwmark segítségével biztosítsa a csomagok elsőbbségét.

3.1. A HTB alkalmazása a kimenő forgalom visszafogására

Bár az útválasztó és a modem közti kapcsolat 10 Mbit/s sebességű, a modem csak 128 kbit/s sebességgel tud adatokat küldeni. Minden ezt a rátát meghaladó sebességű csomag várakozósorba áll a modemnél. Ezért egy ping csomag, amit az útválasztóról küldünk, azonnal elmehet a modemhez, de néhány másodpercet vehet igénybe, amíg ténylegesen kikerül az Internetre, ha a modem várakozósora tartalmaz már valamennyi csomagot. Sajnos a legtöbb ADSL modem esetében nem adhatjuk meg a várakozósor kezelésének módját illetve annak nagyságát. Így először a kimenő csomagokat átirányítjuk oda, ahol mindezt megtehetjük.

Ezt a HTB sorral valósítjuk meg, így csökkentve az ADSL modemhez küldött csomagok számát. Még ha a kifelé menő sávszélességünk a 128kbit/s is lehetne, kicsivel ez alá korlátozzuk le a csomagküldés mértékét. Ha csökkenteni akarjuk a lappangási időt, BIZONYOSNAK kell lennünk, hogy soha egyetlen csomag sem áll sorba a modemnél. Kísérletezések során azt találtam, hogy a kimenő forgalom körülbelül 90kbit/s-ra való visszavételével a sávszélesség majdnem 95%-át tudom elérni a HTB vezérlés nélkül. A HTB engedélyezésével ennél a mértéknél kivédjük, hogy a modem várakozósorba rakja a csomagokat.

3.2. Prioritásos várakozósor kialakítása HTB-vel

> **Megjegyzés:** Megjegyzés: az ebben a részben lévő előző igények (eredetileg N-sávos prioritásos várakozósor kialakításának hívták) később hibásnak találtattak. LEHETETLEN volt a várakozósor különböző sávjaiba tartozó csomagok megjelölése csak a fwmark mező által, viszont ezt gyengén dokumentáltuk a dokumentum 0.1-es verziójának írásakor.

Ennél a pontnál még nem veszünk észre semmi változást a teljesítményben. Pusztán csak áthelyezzük a FIFO sort az ADSL modemtől az útválasztóhoz. Valójában, a Linux alapértelmezésben beállított 100 csomag méretű FIFO sorával, valószínűleg még rosszabbá is tettük a helyzetünket! De nem sokáig...

Minden szomszédos osztálynak adhatunk egy prioritást a HTB soron belül. A különböző típusú forgalom különböző osztályokba helyezésével, majd ezen osztályokhoz különböző prioritások csatolásával, vezérelhetjük a csomagok várakozási sorból való kivételét és elküldését. A HTB lehetővé teszi ezt, miközben megakadályozza egy osztály "kiéheztetését", mivel megadhatjuk a minimálisan garantált mértéket minden osztály számára. Ezenfelül a HTB megengedi azt is, hogy megadjuk egy osztálynak: használhatja másik osztályok nem használt sávszélességét, egy bizonyos felső határig.

Miután beállítottuk az osztályainkat, szűrőket helyezünk el, hogy a forgalmat elhelyezzük beléjük. Több útja is van ennek, de az itt leírt módszer az ismerős iptables/ipchains-t használja a csomagok fwmark-al (tűzfal jelölése a csomagon) történő megjelölésére. A szűrők a csomagok fwmark-ját figyelembe véve helyezik el a forgalmat a HTB sor osztályaiba. Ezen a módon képesek leszünk megfelelő szabályok felállítására az iptables-en belül, hogy bizonyos típusú forgalmat egy bizonyos osztályba küldjön.

3.3. A kimenő csomagok osztályozása iptables-szel

> **Megjegyzés:** eredetileg a dokumentumban az ipchains-t használtuk a csomagok besorolására. Most az újabb iptables-t használjuk.

Az utolsó lépés ahhoz, hogy az útválasztó elsőbbséget adjon az interaktív forgalomnak - a tűzfal beállítása: adjuk meg a forgalom besorolásának módját. Ez a csomag fwmark mezőjének beállításával érhető el.

Anélkül, hogy túlzottan a részletekbe merülnénk, álljon itt az egyszerűsített leírása annak, hogyan lehet a kimenő csomagokat 4 osztályba sorolni úgy, hogy a legmagasabb prioritású lesz a 0x00:

1. Jelöljünk MINDEN csomagot 0x03-al. Ez alapértelmezésben minden csomagot a legalacsonyabb prioritású sorba helyez el.
2. Jelöljük az ICMP csomagokat 0x00-al. Szeretnénk, ha a ping mutatná a legmagasabb prioritású csomagok várakozási idejét.
3. Jelöljünk minden csomagot, aminek célportja 1024 vagy kisebb, 0x01-el. Ez elsőbbséget biztosít az olyan rendszerszolgáltatásoknak, mint a Telnet és SSH. Az FTP portja szintén ebbe a körbe esik, de az FTP adatforgalom a magasabb portokon helyezkedik el és marad a 0x03 sávban.
4. Jelöljünk minden csomagot, aminek a célportja 25 (SMTP), a 0x03-al. Ha valaki levelet fog küldeni egy nagy csatolt állománnyal, nem akarjuk, hogy elárassza az interaktív forgalmat.

5. Jelöljünk minden csomagot, aminek célja egy többszereplős játék-szerver, 0x02-vel. Ez a játékosoknak alacsony lappangási időt biztosít, de megakadályozza, hogy elfoglalják az alacsony várakozást igénylő rendszerszolgáltatásokat.

   Jelöljünk minden "kicsi" csomagot 0x02-vel. A kimenő ACK csomagokat a befelé irányuló letöltésekből azonnal ki kell küldenünk, hogy biztosítsuk a megfelelő letöltéseket. Ez az iptables "length" moduljával lehetséges.

Természetesen ezeket a kívánalmaknak megfelelően átalakíthatod.

3.4. Még néhány fogás...

Két további dolgot tehetsz a lappangási idő javítására. Először is, a maximális átvihető egység (MTU) méretét kisebbre veheted, mint az alapértelmezett 1500 bájt. Ennek a számnak a csökkentése egyben az átlagos időt is csökkenti, amit az elsőbbséget élvező csomagok elküldésére kell fordítani, ha már egy teljes méretű alacsony prioritású csomag küldése folyamatban van. Ennek az értéknek a csökkentése kicsit csökkenti a teljesítményt is, mivel minden csomag legalább 40 bájtnyi IP és TCP fejléc-információt tartalmaz.

A másik dolog a javításhoz, még alacsony prioritású forgalom esetén is, hogy csökkented a várakozási sor méretét az alapértelmezett 100-ról, ami egy ADSL vonalon 10 másodperc alatt ürül ki egy 1500 bájtos MTU-t alapul véve.

3.5. Próbálkozás a bejövő forgalom visszafogására

A "közbenső várakozósor-eszköz" (Intermediate Queuing Device, IMQ) használatával az összes bejövő csomagot ugyanúgy egy várakozósoron futtathatjuk át, mint amilyen módon a kimenőket is. A csomagok prioritása ebben az esetben jóval egyszerűbb. Mivel csak a bejövő TCP forgalmat (próbáljuk meg) vezérelni, az összes nem-TCP forgalmat a 0x00 osztályba rakjuk, az összes TCP forgalmat pedig a 0x01 osztályba. A "kis" TCP csomagokat szintén a 0x00 osztályba soroljuk, mert ezek nagy valószínűséggel a már elküldött kimenő adatok ACK csomagjai. Egy standard FIFO sort állítunk be a 0x00 osztályhoz, illetve egy "véletlenszerű korai eldobás" (Random Early Drop, RED) várósort a 0x01 osztályhoz. A RED jobb a FIFO-nál a TCP vezérlését tekintve, mert eldobja a csomagokat már a sor olyan túlcsordulása előtt, mikor megpróbálja lelassítani a forgalmat az ellenőrzés fenntartása érdekében. Ezen kívül mindkét osztályt le fogjuk korlátozni egy maximális bejövő forgalmi határra, ami kisebb, mint a valós, ADSL modemen bejövő sebesség.

3.5.1. Miért nem olyan jó a bejövő forgalom korlátozása?

Korlátozni szeretnénk a bejövő forgalmunkat, hogy elkerüljük a várakozósor betelését a szolgáltatónál, ami néha 5 másodpernyi adat pufferelését jelentheti. A probléma abban áll, hogy jelenleg a bejövő TCP forgalom korlátozásának egyetlen módja a teljesen jó csomagok eldobálása. Ezek a csomagok már foglaltak némi sávszélességet az ADSL modemen, csak a Linux gép dobta el őket abból a célból, hogy a jövőbeni csomagokat lelassítsa. Ezek az eldobott csomagok időnként újra elküldésre kerülnek, még több sávszélességet foglalva. Amikor korlátozzuk a forgalmat, korlátozzuk azon csomagok mértékét, amiket elfogadunk a hálózatunk számára. Mivel az *aktuális* bejövő adatráta valahol efölött van az eldobott csomagok miatt, a bejövő águnkat *sokkal* jobban le kell korlátoznunk az ADSL modem aktuális rátájánál, az alacsony lappangási idő biztosítása érdekében. A gyakorlatban az én 1.5Mbit/s bejövő ágamat 700kbit/s-re kellett korlátoznom, hogy elfogadható szinten tartsam a lappangást 5 egyidejű letöltésnél. Minél több TCP folyamatod van, annál több sávszélességet vesztesz az eldobott csomagok miatt, és annál kisebbre kell venned a korlátozás mértékét.

A bejövő TCP forgalom ellenőrzésének sokkal jobb módja a TCP ablak manipulációja, de ebben a pillanatban nincs (szabadon elérhető) megvalósítása ennek Linuxra (amennyire én tudom...).

4. Megvalósítás

Mindezen okfejtés után most már ideje, hogy megvalósítsuk a sávszélesség-gazdálkodást Linuxon.

4.1. Kikötések

A DSL modemhez aktuálisan küldött adatok mértékének korlátozása nem olyan egyszerű, mint amilyennek látszik. A legtöbb DSL modem igazából csak egy ethernet híd, amik továbbítják az adatokat oda-vissza a Linux gép és a szolgáltatónál lévő gateway között. A legtöbb DSL modem ATM-et használ adatátviteli csatolófelületként. Az ATM mindig 53 bájtos cellákban küldi az adatokat. Ezekből 5 bájt a fejléc információ, és 48 bájt marad az adatoknak. Még ha csak 1 bájt adatot küldesz is, a teljes 53 bájt sávszélességet foglal, mivel az ATM cellák mindig 53 bájt hosszúak. Ez azt jelenti, hogy ha egy tipikus TCP ACK csomagot küldesz, ami 0 bájt adatot + 20 bájt TCP fejlécet + 20 bájt IP fejlécet + 18 bájt Ethernet fejlécet tartalmaz. Valójában, még ha a kiküldött ethernet csomagnak csak 40 bájtnyi "hasznos terhe" van is (TCP és IP fejléc), a legkisebb méret egy Ethernet csomagnál 46 bájtnyi adat, így a maradék 6 bájt 0-val töltődik ki. Ez azt jelenti, hogy az Ethernet csomag plusz a fejléc információk aktuális hossza 18 + 46 = 64 bájt. Az ATM-mel 64 bájt átküldéséhez két ATM cellát kell küldened, ami 106 bájt sávszélességet foglal. Vagyis minden TCP ACK csomagnál 42 bájt sávszélességet vesztesz. Ez rendben van, ha a Linux figyelembe veszi a DSL modem által használt csomag-beágyazást, de ehelyett a Linux csak a TCP és IP fejlécet és 14 bájtos MAC címet jegyzi (a Linux nem számolja a 4 bájtos CRC-t, mivel ezt a hardver szint kezeli). A Linux nem számol a 46 bájtos minimális Ethernet csomagmérettel, sem a fix méretű ATM cellával.

Mindez azt jelenti, hogy a kimenő sávszélességet valamivel kisebbre kell állítani, mint a valós kapacitás (amíg nem találunk egy csomag-időzítőt, ami jegyzi a különböző típusú csomag-beágyazásokat). Azt találhatod, hogy sikerült egy jó értékre beállítani a sávszélességet, de amikor egy nagy fájlt töltesz le, a lappangás felszökik 3 másodperc fölé. Ez legvalószínűbben amiatt van, mivel a Linux rosszul számítja ki a bizonyos kis ACK csomagok által igényelt sávszélességet.

Néhány hónapot dolgoztam ennek a problémának a megoldásán, és majdnem lezártam a dolgot egy olyan megoldással, amit hamarosan közreadok további tesztelésre. A megoldás egy felhasználói szintű várakozósor használatát mutatja be a Linux QoS-e helyett a csomagok korlátozására. Alapvetően egy egyszerű HTB sort alkalmaztam, ami a Linux felhasználói szintű sorait használja. Ez a megoldás (eddig) képes volt a kimenő forgalom OLYAN JÓ korlátozására, hogy még egy masszív letöltés (több szálon) és ugyanilyen feltöltés (gnutella, több szálon) alatt is, a lappangás 400 ms CSÚCSÉRTÉKET ért csak el a névleges, forgalom nélküli 15 ms-hoz képest. További információért erről a QoS módszerről, iratkozz fel a frissítések levelezőlistájára vagy később nézd meg ennek a HOGYANnak a frissebb változatait.

4.2. Szkript: myshaper

A következőkben egy általam a Linux útválasztón a sávszélesség korlátozására használt szkript listája található. Ez több, a dokumentumban foglalt koncepciót is felhasznál. A kimenő forgalom a 7 típustól függő várósor egyikébe kerül. A bejövő forgalom két sorba kerül, a TCP csomagokat (alacsonyabb prioritásúak) előbb eldobjuk, ha a bejövő adatok a mérték fölöttiek. A szkriptben megadott ráták úgy tűnik, jól működnek az én beállításomban, de az eredmények változhatnak.

A szkript eredetileg az ADSL WonderShaper-en alapult, amint megtalálható a LARTC webhelyen (http://www.lartc.org).

```
#!/bin/bash
#
# myshaper - DSL/kabelmodem kimeno forgalmanak szabalyozasa.
#            Az ADSL/Cable wondershaper (www.lartc.org) szkripten alapszik.
#
# Irta: Dan Singletary (8/7/02)
#
# FIGYELEM: a szkript feltetelezi,  hogy a kernelt megfoltoztuk a megfelelo HTB
# sor és  IMQ foltokkal,  amik hozzaferhetok  itt (megj.:  az ujabb kerneleknel
# lehet, hogy nem kell folt):
#
#       http://luxik.cdi.cz/~devik/qos/htb/
#       http://luxik.cdi.cz/~patrick/imq/
#
# Konfiguracios beallitasok:
#  DEV    - ethX-re allitsuk, ami kapcsolodik a DSL/kabelmodemhez
#  RATEUP - allitsuk valamivel kisebbre, mint a modem kimeno savszelessege.
#           Nekem 1500/128  DSL vonalam  van, es a  RATEUP=90 jol mukodik a
#           128 kbps-os feltoltessel. De ahogy jonak latod.
#  RATEDN - allitsd valamivel kisebbre, mint a bejovo savszelesseg.
#
#
# Teoria az imq hasznalatarol a bejovo forgalom alakitasahoz:
#
#
# BEJOVO  TCP  KAPCSOLATOK  BEFOLYASOLASAT.  Ennek  ertelmeben  minden   nem-TCP
# forgalmat egy  magas prioritasu  osztalyba kell  sorolnunk, mivel  egy nem-TCP
# csomag eldobasa valoszinuleg a csomag  ujrakuldeset okozza. Ez semmi mast  nem
# jelent,  csak  a  savszelesseg  szuksegtelen  lefoglalasat,  hogy specifikusan
# valaszthatunk:  NEM  dobunk  el  bizonyos  tipusu  csomagokat, amiket magasabb
# prioritasu tarolokba  helyezunk el  (ssh, telnet  stb). Ez  azert van,  mert a
# csomagok  mindig  az  alacsonyabb  prioritasu  osztalybol  jonnek  elo azzal a
# kikotessel,  hogy  a  csomagok  meg  minden osztalybol egyforman egy minimalis
# mertekben jonnek ki (ebben a szkriptben minden tarolo legalabb a  tisztesseges
# 1/7 savszelessegnyivel A TCP csomag  eldobasa egy kapcsolaton belul a  fogadas
# alacsonyabb mertekehez vezet, a torlodas-elkerulo algoritmus miatt.
#
#     *  Semmit  nem  nyerunk  a  nem-TCP  csomagok  eldobasaval.  Valojaban, ha
#     fontosak  voltak,  ugyis  ujra  elkuldik  oket, igy megprobaljuk azt, hogy
#     sosem dobjuk el oket. Ez azt jelenti, hogy a telitett TCP kapcsolatok  nem
#     befolyasoljak  negativan  azokat  a  protokollokat,  amelyeknel  nincs   a
#     TCP-hez hasonlo beepitett ujrakuldes.
#
#     * A TCP kapcsolatok lelassitasa  ugy, hogy a teljes bejovo  rata kevesebb,
#     mint az eszkoz  valos kapacitasa AZT  OKOZHATJA, hogy keves  vagy egyetlen
#     csomag   sem   all   varakozosorba   a   szolgaltatoi   oldalon    (DSLAM,
#     kabel-koncentrator  stb).   Mivel  ezek  a  sorok  kepesek  megtartani   4
#     masodpercnyi adatot 1500Kbps sebessegen  vagy 6 megabitnyi adatot,  ha egy
```

```
#     csomag sem all sorba, az alacsonyabb lappangast okoz.
#
# Kikotesek (kerdesfeltevesek a teszteles elott):
# * A bejovo forgalom ezen a modon valo korlatozasa gyenge TCP-teljesítmenyt ad?
# - Az elozetes valsz: nem! Ugy nez ki, hogy az ACK csomagok prioritasanak
# beallitasa (kicsi <64b) anelkul  maximaljuk a kimeno telesitmenyt,  hogy
# nem  vesztunk  savszelesseget a mar meglevo ujrakuldott  csomagok miatt.

# Megjegyzes: a kovetkezo konfiguracio jol mukodik az en beallitasaimmal:
# 1.5M/128K ADSL a Pacific Bell Internet-en keresztul (SBC Global Services)

DEV=eth0
RATEUP=90
RATEDN=700   # Figyeld meg, hogy ez jelntosen kisebb mint az 1500-as kapacitas.
# Emiatt nem kell a bejovo  forgalom korlatozasaval torodnod, amig
# nem hasznalhatunk  jobb megvalositast,  mint például a TCP ablak
# manipulacioja.
#
# konfiguracios beallitasok vege
#

if [ "$1" = "status" ]
then
        echo "[qdisc]"
        tc -s qdisc show dev $DEV
        tc -s qdisc show dev imq0
        echo "[class]"
        tc -s class show dev $DEV
        tc -s class show dev imq0
        echo "[filter]"
        tc -s filter show dev $DEV
        tc -s filter show dev imq0
        echo "[iptables]"
        iptables -t mangle -L MYSHAPER-OUT -v -x 2> /dev/null
        iptables -t mangle -L MYSHAPER-IN -v -x 2> /dev/null
        exit
fi

# Mindent visszaalitunk alapallapotba (torlunk)
tc qdisc del dev $DEV root    2> /dev/null > /dev/null
tc qdisc del dev imq0 root 2> /dev/null > /dev/null
iptables -t mangle -D POSTROUTING -o $DEV -j MYSHAPER-OUT 2> /dev/null > /dev/null
iptables -t mangle -F MYSHAPER-OUT 2> /dev/null > /dev/null
iptables -t mangle -X MYSHAPER-OUT 2> /dev/null > /dev/null
iptables -t mangle -D PREROUTING -i $DEV -j MYSHAPER-IN 2> /dev/null > /dev/null
iptables -t mangle -F MYSHAPER-IN 2> /dev/null > /dev/null
iptables -t mangle -X MYSHAPER-IN 2> /dev/null > /dev/null
ip link set imq0 down 2> /dev/null > /dev/null
rmmod imq 2> /dev/null > /dev/null

if [ "$1" = "stop" ]
then
        echo "Shaping removed on $DEV."
```

```
        exit
fi

############################################################
#
# Kimeno korlatozas (a teljes savszelesseg RATEUP-ra allitva)

# a varakozosor meretet ugy allitjuk be, hogy kb. 2 mp lappangas legyen az alacsony
# prioritasu csomagoknal
ip link set dev $DEV qlen 30

# a kimeno eszkozon MTU-t allitunk. Az MTU csokkentese alacsonyabb lappangast
# ad, de valamivel kisebb kimeno teljesitmenyt is az IP es TCP protokoll
# felulvezerlese miatt

ip link set dev $DEV mtu 1000

# a HTB-t gyoker qdisc-nek allitjuk be
tc qdisc add dev $DEV root handle 1: htb default 26

# hozzaadjuk a fobb korlatozo osztalyokat
tc class add dev $DEV parent 1: classid 1:1 htb rate ${RATEUP}kbit

# hozzadjuk az alosztalyokat - garantaljuk minden osztalynak LEGALABB a
# "tisztesseges" osztozast a savszelessegen. Emiatt egy osztalyt sem fog egy
# masik kieheztetni. Ezenkivul mindegyik osztaly hasznalhatja a rendelkezesre
# allo savszelesseget, ha a tobbi nem hasznalja.

tc class add dev $DEV parent 1:1 classid 1:20 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 0
tc class add dev $DEV parent 1:1 classid 1:21 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 1
tc class add dev $DEV parent 1:1 classid 1:22 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 2
tc class add dev $DEV parent 1:1 classid 1:23 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 3
tc class add dev $DEV parent 1:1 classid 1:24 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 4
tc class add dev $DEV parent 1:1 classid 1:25 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 5
tc class add dev $DEV parent 1:1 classid 1:26 htb rate $[$RATEUP/7]kbit ceil ${RATEUP}kbit prio 6


# az alosztalyokhoz qdisc-eket adunk - SFQ-t adunk minden osztalyhoz. Az SFQ
# biztositja, hogy minden osztalyon belul a kapcsolatokat (majdnem) egyenloen
# kezeljuk.


tc qdisc add dev $DEV parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev $DEV parent 1:21 handle 21: sfq perturb 10
tc qdisc add dev $DEV parent 1:22 handle 22: sfq perturb 10
tc qdisc add dev $DEV parent 1:23 handle 23: sfq perturb 10
tc qdisc add dev $DEV parent 1:24 handle 24: sfq perturb 10
tc qdisc add dev $DEV parent 1:25 handle 25: sfq perturb 10
tc qdisc add dev $DEV parent 1:26 handle 26: sfq perturb 10

# az fwmark-kal szurjuk osztalyokba a forgalmat - itt a csomagon beallitott
# fwmark-nak megfeleloen iranyitjuk a forgalmat az osztalyokba (az fwmark-ot az
# iptables segitsegevel kesobb allitjuk be). Figyeld meg, hogy fentebb az
```

```
# alapertelmezett  prioritasu  osztalyt  1:26-ra  allitottuk,  igy  a nem jelolt
# csomagok (vagy a nem  ismert ID-ju csomagok) alapertelmezesben  az alacsonyabb
# prioritasu osztalyba mennek.

tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 20 fw flowid 1:20
tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 21 fw flowid 1:21
tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 22 fw flowid 1:22
tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 23 fw flowid 1:23
tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 24 fw flowid 1:24
tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 25 fw flowid 1:25
tc filter add dev $DEV parent 1:0 prio 0 protocol ip handle 26 fw flowid 1:26

#
# a MYSHAPER-OUT lanc hozzadasa az  iptables "mangle" tablajahoz - ez beallitja
# azt a tablat, amit a csomagok szuresehez es megjelolesehez hasznalunk.

iptables -t mangle -N MYSHAPER-OUT
iptables -t mangle -I POSTROUTING -o $DEV -j MYSHAPER-OUT

# a fwmark ertekek  beallitasa a kulonbozo  tipusu forgalomhoz - a  fwmark-ot 20-26
# kozottire allitjuk a kivant osztalynak megfeleloen. A 20 a legmagasabb prioritas.

iptables -t mangle -A MYSHAPER-OUT -p tcp --sport 0:1024 -j MARK --set-mark 23 # Alapertek az
# alacsony portokon zajlo forgalomhoz
iptables -t mangle -A MYSHAPER-OUT -p tcp --dport 0:1024 -j MARK --set-mark 23 # ""
iptables -t mangle -A MYSHAPER-OUT -p tcp --dport 20 -j MARK --set-mark 26     # ftp-data port, alac
iptables -t mangle -A MYSHAPER-OUT -p tcp --dport 5190 -j MARK --set-mark 23   # aol instant messeng
iptables -t mangle -A MYSHAPER-OUT -p icmp -j MARK --set-mark 20               # ICMP (ping) - magas
iptables -t mangle -A MYSHAPER-OUT -p udp -j MARK --set-mark 21                # DNS nevfeloldas (ki
iptables -t mangle -A MYSHAPER-OUT -p tcp --dport ssh -j MARK --set-mark 22    # secure shell
iptables -t mangle -A MYSHAPER-OUT -p tcp --sport ssh -j MARK --set-mark 22    # secure shell
iptables -t mangle -A MYSHAPER-OUT -p tcp --dport telnet -j MARK --set-mark 22 # telnet (ew...)
iptables -t mangle -A MYSHAPER-OUT -p tcp --sport telnet -j MARK --set-mark 22 # telnet (ew...)
iptables -t mangle -A MYSHAPER-OUT -p ipv6-crypt -j MARK --set-mark 24         # IPSec - viszont nem
iptables -t mangle -A MYSHAPER-OUT -p tcp --sport http -j MARK --set-mark 25   # helyi webszerver
iptables -t mangle -A MYSHAPER-OUT -p tcp -m length --length :64 -j MARK --set-mark 21 # kis csomago
iptables -t mangle -A MYSHAPER-OUT -m mark --mark 0 -j MARK --set-mark 26      # redundans- jeloljun

# vegeztunk a kimeno korlatozassal
#
#########################################################

echo "Outbound shaping added to $DEV.  Rate: ${RATEUP}Kbit/sec."

# tavolitsd el a megjegyzest a kovetkezo sor elol, ha csak kimeno forgalomszabalyozast akarsz
# exit

#########################################################
#
# Bejovo korlatozas (a teljes savszelesseg RATEDN-re allitva)

# megnezzuk, hogy az imq modul betoltodott-e
```

```
modprobe imq numdevs=1

ip link set imq0 up

# a qdisc hozzadasa - alapertelmezett alcsony prioritasu 1:21-es osztaly

tc qdisc add dev imq0 handle 1: root htb default 21

# a fo korlatozo osztalyok hozzaadasa
tc class add dev imq0 parent 1: classid 1:1 htb rate ${RATEDN}kbit

# alosztalyok hozzaadasa - TCP forgalom a 21-ben, nem-TCP forgalom a 20-ban
#
tc class add dev imq0 parent 1:1 classid 1:20 htb rate $[$RATEDN/2]kbit ceil ${RATEDN}kbit prio 0
tc class add dev imq0 parent 1:1 classid 1:21 htb rate $[$RATEDN/2]kbit ceil ${RATEDN}kbit prio 1

# az alosztalyokhoz qdisc-eket  adunk - SFQ-t adunk minden osztalyhoz. Az SFQ
# biztositja, hogy minden osztalyon belul a kapcsolatokat (majdnem) egyenloen
# kezeljuk.

tc qdisc add dev imq0 parent 1:20 handle 20: sfq perturb 10
tc qdisc add dev imq0 parent 1:21 handle 21: red limit 1000000 min 5000 max 100000 avpkt 1000 burst

# az fwmark-kal  szurjuk osztalyokba  a forgalmat  - itt  a csomagon  beallitott
# fwmark-nak megfeleloen iranyitjuk a forgalmat az osztalyokba (az fwmark-ot  az
# iptables  segitsegevel  kesobb  allitjuk  be).  Figyeld  meg,  hogy fentebb az
# alapertelmezett  prioritasu  osztalyt  1:21-re  allitottuk,  igy  a nem jelolt
# csomagok (vagy a nem  ismert ID-ju csomagok) alapertelmezesben  az alacsonyabb
# prioritasu osztalyba kerulnek.


tc filter add dev imq0 parent 1:0 prio 0 protocol ip handle 20 fw flowid 1:20
tc filter add dev imq0 parent 1:0 prio 0 protocol ip handle 21 fw flowid 1:21


# a MYSHAPER-IN lanc hozzadasa az iptables "mangle" tablajahoz - ez beallitja azt a tablat,
# amit a csomagok szuresehez es megjelolesehez hasznalunk.

iptables -t mangle -N MYSHAPER-IN
iptables -t mangle -I PREROUTING -i $DEV -j MYSHAPER-IN

# a fwmark ertekek beallitasa a kulonbozo tipusu forgalomhoz - a fwmark-ot 20-26 kozottire
# allitjuk a kivant osztalynak megfeleloen. A 20 a legmagasabb prioritas.


iptables -t mangle -A MYSHAPER-IN -p ! tcp -j MARK --set-mark 20             # A nem-tcp csomagokat
iptables -t mangle -A MYSHAPER-IN -p tcp -m length --length :64 -j MARK --set-mark 20 # rovid TCP cs
iptables -t mangle -A MYSHAPER-IN -p tcp --dport ssh -j MARK --set-mark 20    # secure shell
iptables -t mangle -A MYSHAPER-IN -p tcp --sport ssh -j MARK --set-mark 20    # secure shell
iptables -t mangle -A MYSHAPER-IN -p tcp --dport telnet -j MARK --set-mark 20 # telnet (ew...)
iptables -t mangle -A MYSHAPER-IN -p tcp --sport telnet -j MARK --set-mark 20 # telnet (ew...)
iptables -t mangle -A MYSHAPER-IN -m mark --mark 0 -j MARK --set-mark 21             # redundans- m
```

```
# vegul utasitjuk ezeket a csomagokat, hogy menjenek keresztul a fent beallitott imq0-on

iptables -t mangle -A MYSHAPER-IN -j IMQ

# vegeztunk a bejovo forgalommal
#
######################################################

echo "Inbound shaping added to $DEV.  Rate: ${RATEDN}Kbit/sec."
```

5. Az új várakozósor tesztelése

A legkönnyebben azzal tesztelheted az új beállítást, hogy telíted a felfelé irányuló ágat alacsony prioritású forgalommal. Ez a prioritások beállításától függ. A példa kedvéért, mondjuk a telnet és a ping forgalmat helyezted magasabb prioritásba (alacsonyabb fwmark), a többi magasabb portot (amik FTP átvitelhez stb. használatosak) pedig alacsonyabba. Ha indítasz egy FTP feltöltést a kifelé menő sávszélesség telítésére, csak a gateway felé (a DSL vonal másik felén lévő) menő ping idők kis mértékű növekedését tapasztalhatod, összehasonlítva a prioritásos várósor nélküli értékekkel. A 100 ms alatti ping idők tipikusak attól függően, hogyan állítottad be a dolgokat. Az egy vagy két másodpercnél nagyobb idők valószínűleg az jelzik, hogy nem működnek rendben a dolgok.

6. Rendben, működik! Hogyan tovább?

Most, hogy sikeresen elkezdtél gazdálkodni a sávszélességeddel, elgondolkodhatsz azon, hogyan használod ki. Végül is, valószínűleg fizetsz érte!

- Gnutella kliens használata és FÁJLJAID MEGOSZTÁSA a hálózat teljesítményének kedvezőtlen befolyásolása nélkül.
- Web szervert futtathatsz anélkül, hogy a weblapok kiszolgálása lelassítaná a Quake partit.

7. Kapcsolódó hivatkozások

- Bandwidth Controller for Windows - http://www.bandwidthcontroller.com
- dsl_qos_queue (http://www.sonicspike.net/software#dsl_qos_queue) - (béta) Linuxhoz. Nincs kernel-foltozás, és jobb a teljesítmény.