ADSL Bandwidth Management HOWTO

Dan Singletary

dvsing@sonicspike.net

Diario delle Revisioni

Revisione 1.3 2003-04-07 Revisionato da: ds
Aggiunta la sezione Link correlati.
Revisione 1.2 2002-09-26 Revisionato da: ds
Aggiunto il link alla nuova lista di discussione via email. Aggiunta un'informazione nella sezione riguardante il nuovo e migl
Revisione 1.1 2002-08-26 Revisionato da: ds
Alcune correzioni (grazie alle molte persone che le hanno fatte notare!). Aggiunta un'informazione alla sezione sull'implem
Revisione 1.0 2002-08-21 Revisionato da: ds
Miglior controllo sulla banda, più teoria, aggiornato per i kernel 2.4.
Revisione 0.1 2001-08-06 Revisionato da: ds
Pubblicazione iniziale

Questo documento descrive come configurare un router Linux per controllare piu efficacemente il traffico in uscita di un modem ADSL o altro dispositivo con stesse proprietà di larghezza di banda (cable modem, ISDN ecc.). Viene messa l'enfasi sull'abbassamento della latenza per il traffico interattivo anche quando la larghezza di banda in upstream e/o in downstream è completamente satura. Traduzione a cura di Michele Ferritto, ferritto@toglimi.libero.it, e revisione a cura di Sandro Cardelli.

Sommario

1. Introduzione

Lo scopo di questo documento è suggerire un modo per controllare il traffico in uscita su una connessione ADSL (o cable modem) a Internet. Il problema è che molte linee ADSL sono limitate nelle vicinanze dei 128kbps per i trasferimenti upstream. Ad aggravare questo c'è la coda dei pacchetti nel modem ADSL, la quale può impiegare da 2 a 3 secondi per svuotarsi quando è piena. Ciò significa che quando la larghezza di banda in upstream è completamente satura possono passare fino a 3 secondi prima che qualsiasi altro nuovo pacchetto di dati possa uscire verso Internet, mettendo in crisi applicazioni interattive quali telnet e giochi in modalità multi-player.

1.1. Nuove versioni di questo documento

Si veda, per l'ultima versione di questo documento, sul World Wide Web presso l'URL: http://www.tldp.org.

Nuove versioni di questo documento saranno trasferite ai vari siti Linux WWW e FTP, inclusa l'home page di LDP presso http://www.tldp.org.

1.2. Liste di discussione

Per domande e informazioni a proposito dell'ADSL Bandwidth Management per favore iscrivetevi all'ADSL Bandwidth Management mailing list presso http://jared.sonicspike.net/mailman/listinfo/adsl-qos.

1.3. Liberatoria

Ne l'autore ne i distributori o alcuno dei contributori di questo HOWTO sono in nessun modo responsabili per danni fisici, finanziari, morali o di altro tipo che dovessero occorrere seguendo i suggerimenti di questo testo.

1.4. Copyright and License

1.5. Copyright e Licenza d'uso

1.6. Feedback e correzioni

Se avete domande o commenti da fare su questo documento, sentitevi liberi di contattare l'autore presso dvsing@sonicspike.net (mailto:dvsing@sonicspike.net).

2. Background

2.1. Prerequisiti

Il metodo descritto in questo documento dovrebbe funzionare anche in altri scenari in ogni caso rimane non testato in tutte le configurazioni eccettuata la seguente:

- Red Hat Linux 7.3
- Kernel 2.4.18-5 con il supporto QoS fully enabled (moduli OK) con incluse le seguenti patch (le quali potrebbero essere incluse in kernel più recenti):
  - HTB queue - http://luxik.cdi.cz/~devik/qos/htb/

    Nota: è stato segnalato che i kernel dalla versione 2.4.18-3 distibuiti con Mandrake (8.1, 8.2) sono già patchati per l'HTB.
  - IMQ device - http://luxik.cdi.cz/~patrick/imq/
- iptables 1.2.6a o superiore (la versione di iptables distribuita con Red Hat 7.3 manca del length module)

> Nota: Precedenti versioni di questo documento specificano un metodo di controllo di banda che richiede di patchare l'esistente queue sch_prio. È stato scoperto successivamente che questa patch non era necessaria. A parte ciò il nuovo metodo descritto in questo documento vi darà migliori risultati (sebbene durante la stesura di questo documento siano necessarie *2* patch del kernel. :) Buon patching.)

2.2. Schema

Per facilitare le cose, tutti i riferimenti ai dispositivi di rete e le configurazioni in questo documento si riferiranno al seguente schema:

```
             <-- 128kbit/s      --------------     <-- 10Mbit -->
  Internet <---------------------> | ADSL Modem | <--------------------
              1.5Mbit/s -->     --------------                     |
                                                                   | eth0
                                                                   V
                                                      -----------------
                                                      |               |
                                                      | Linux Router  |
                                                      |               |
                                                      -----------------
                                                       | .. | eth1..ethN
                                                       |    |
                                                       V    V

                                                    Rete Locale
```

2.3. Le Code Pacchetto(Packet Queues)

Le Code Pacchetto sono contenitori che mantengono i dati per un dispositivo di rete quando essi non possono essere immediatamente inviati. La maggior parte delle Code Pacchetto utilizzano una disciplina FIFO (first in, first out)a meno che non siano state configurate per fare altrimenti. Questo significa che quando la Coda Pacchetto di un particolare device è completamente piena, il pacchetto più recente posto in essa sarà inviato al dispositivo solo dopo che tutti gli altri pacchetti nella coda in quel momento saranno stati inviati.

2.3.1. L'Upstream

Con un modem ADSL, la larghezza di banda è asimmetrica con 1.5Mbit/s di downstream e 128kbit/sec di upstream tipici. Benché questa sia la velocità della linea, l'interfaccia tra il Router Linux e il modem ADSL è tipicamente prossima ai 10Mbit/s. Se l'interfaccia con la rete locale è lo stesso a 10Mbit/s, ci sarà tipicamente un NO QUEUING sul router quando i pacchetti vanno dalla Rete Locale a Internet. I pacchetti sono inviati fuori da eth0 alla stessa velocità a cui sono stati ricevuti dalla Rete Locale. Diversamente sul modem ADSL, i pacchetti vengono messi in coda, dato che essi arrivano a 10Mbit/s e possono essere inviati solo a 128kbit/s. Nell'eventualità la Coda Pacchetto sul modem ADSL diventa piena e ogni pacchetto in più inviatogli verrà silenziosamente scartato. Il TCP è progettato per ovviare a questo problema e provvederà ad aggiustare la dimensione della finestra di trasmissione per ottenere il massimo vantaggio dalla banda disponibile.

Mentre le Code Pacchetto combinate con il TCP danno come risultato un più efficiente uso della larghezza di banda, grandi code FIFO possono aumentare la latenza per il traffico di tipo interattivo.

Un altro tipo di coda con una disciplina alquanto simile a FIFO viene detta coda di priorità n-band (n-band priority queue). In questo caso, invece di avere una sola coda dove mettere i dati, la coda di priorità n-band ha n code FIFO nelle quali i pacchetti sono messi a seconda della loro classificazione. Ogni coda ha una priorità e i pacchetti sono sempre inviati a partire dalla coda con priorità più alta che ne contiene. Utilizzando questa disciplina i pacchetti FTP possono essere messi in una coda con priorità più bassa rispetto a pacchetti telnet cosi che durante un FTP upload, un singolo pacchetto telnet può saltare la coda ed essere inviato immediatamente.

Questo documento è stato rivisto per utilizzare una nuova coda di linux chiamata Hierarchical Token Bucket (HTB). La coda HTB è qualcosa in più rispetto alla coda n-band descritta sopra, e ha la capacità di limitare il tasso di traffico in ciascuna classe. In aggiunta, ha anche la possibilità di creare classi di traffico al di sotto di altre classi creando una gerarchia. La descrizione completa di HTB va oltre lo scopo di questo documento, maggiori informazioni si trovano presso http://www.lartc.org

2.3.2. Il Downstream

Il traffico entrante sul vostro modem ADSL viene messo in coda all'incirca come il traffico uscente , in questo caso la coda sta dal vostro ISP. A causa di ciò, probabilmente non avete il controllo diretto di come i pacchetti vengono messi in coda o quale tipo di traffico richiede trattamento preferenziale. L'unico modo di mantenere la latenza bassa è essere sicuri che gli utenti non inviino dati troppo velocemente. Sfortunatamente, non c'è modo di controllare direttamente la velocità a cui i pacchetti arrivano, tuttavia da quando la gran parte del traffico sulle vostre reti è costituita dal TCP, ci sono alcuni modi di rallentare chi invia i dati:

- Scartare intenzionalmente pacchetti entranti - il TCP è progettato per ottenere il massimo vantaggio dalla banda disponibile e allo stesso tempo evitare la congestione del link. Questo significa che durante un trasferimento , TCP invia una sempre maggiore quantità di dati fino a che eventualmente un pacchetto viene scartato. TCP riesce a rilevare questo e di conseguenza riduce la sua finestra di trasmissione. Questo ciclo continua durante tutto il processo e assicura che i dati vengano spostati il più velocemente possibile.

- Manipolare gli avvisi di ricezione finestra - Durante un trasferimento TCP , il ricevente invia indietro un flusso continuo di pacchetti di acknowledgment (ACK). Inclusi in questi pacchetti ACK c'è un avviso di dimensione finestra il quale dichiara il totale massimo di dati non riconosciuti che il ricevente dovrebbe inviare. Manipolando la dimensione finestra dei pacchetti ACK uscenti possiamo intenzionalmente rallentare chi invia i dati. Attualmente non c'è nessuna (libera) implementazione per questo tipo di controllo di flusso su Linux (Comunque potrei lavorarci su!).

3. Come Funziona

Ci sono due passi fondamentali per ottimizzare la larghezza di banda upstream. Per primo dobbiamo trovare un modo per prevenire che il modem ADSL metta in coda pacchetti fino a che non abbiamo il controllo su come esso la tratti. Per fare questo dovremo limitare la quantità di dati che il router invia su eth0 per rimanere leggermente sotto rispetto alla larghezza di banda totale dell'upstream del modem ADSL. Questo risulterà nell'avere il router che mette in coda pacchetti che arrivano dalla Rete Locale più velocemente di quanto esso possa inviarne.

Il secondo passo consiste nell'inserire una priorità di disciplina di coda sul router. Studieremo una coda che può essere configurata per dare priorità al traffico interattivo come il telnet o i giochi in modalità multi-player.

> Utilizzando la coda HTB possiamo realizzare lo shaping della larghezza di banda e la priorità di messa in coda allo stesso tempo assicurando anche nel mentre che nessuna priorità di classe rimanga senza banda. Evitare questo fenomeno non era possibile utilizzando il metodo evidenziato nella revisione 0.1 di questo documento.

Il passo finale è di configurare il firewall per assegnare la priorità ai pacchetti utilizzando il campo fwmark.

3.1. Limitare il traffico uscente con Linux HTB

Sebbene la connessione tra il router e il modem è a 10Mbit/s, il modem è capace di inviare dati solo a 128kbit/s. Ogni dato che ecceda questa quota verrà messo in coda sul modem. In questo modo, un pacchetto ping inviato dal router può andare al modem immediatamente, ma può impiegare qualche secondo per essere inviato effettivamente a Internet se la coda del modem ha qualche pacchetto al suo interno. Sfortunatamente molti modem ADSL non hanno nessun meccanismo per specificare come i pacchetti sono levati dalla coda o quanto sia larga questa, così il nostro primo obbiettivo è spostare il luogo dove i pacchetti in uscita sono messi in coda da qualche parte dove possiamo avere maggior controllo su di essa.

Possiamo realizzare questo utilizzando la coda HTB per limitare il tasso al quale inviamo i pacchetti al modem ADSL. Anche se la nostra larghezza di banda upstream può essere 128kbit/s dovremo limitare il valore al quale inviamo i pacchetti per fare in modo che sia leggermente inferiore a questo. Se vogliamo abbassare la latenza dobbiamo essere SICURI che neanche un singolo pacchetto sia messo in coda sul modem. Tramite prove ho scoperto che limitare il traffico uscente a circa 90kbit/s mi da quasi il 95% della larghezza di banda che posso ottenere senza il controllo di tasso HTB. Con HTB abilitato a questo valore, preveniamo la messa in coda dei pacchetti da parte del modem ADSL.

3.2. Priorità di messa in coda con HTB

□

A questo punto non abbiamo ancora realizzato nessun cambiamento nella performance. Abbiamo semplicemente spostato la coda FIFO dal modem ADSL al router. In effetti, con Linux configurato con una coda di default di 100 pacchetti abbiamo probabilmente peggiorato il nostro problema! Ma non per molto...

Ad ogni classe adiacente in una coda HTB può essere assegnata una priorità. Mettendo differenti tipi di traffico in differenti classi e assegnando a queste classi differenti priorità, possiamo controllare l'ordine con il quale i pacchetti sono levati dalla coda e inviati. HTB rende ciò possibile, evitando contemporaneamente il prosciugamento delle altre classi, poiché c'è la possibilità di specificare un valore minimo garantito per ogni classe. In aggiunta, HTB ci permette di dire ad una particolare classe che può utilizzare qualsiasi larghezza di banda non utilizzata da altre classi fino ad una certa soglia.

Una volta che abbiamo le nostre classi pronte, dobbiamo creare dei filtri per distribuire il traffico nelle suddette. Ci sono diversi modi per farlo, ma il metodo descritto in questo documento utilizza i più noti comandi iptables/ipchains per marcare i pacchetti con un valore fwmark. I filtri mettono il traffico nelle classi della coda HTB basandosi sul loro fwmark. In questo modo, abbiamo la possibilità di creare delle matching rules con iptables per inviare certi tipi di traffico a determinate classi.

3.3. Classificare i pacchetti uscenti con iptables

□

Il passo finale nella configurazione del vostro router per dare priorità al traffico interattivo è creare il firewall per definire come il traffico deve essere classificato. Questo viene ottenuto settando il campo fwmark del pacchetto.

Senza entrare in dettagli, di seguito c'è una descrizione semplificata di come i pacchetti uscenti possono essere classificati in 4 categorie con la 0x00 avente la priorità più alta:

1. Marcare TUTTI i pacchetti come 0x03. Questo li posiziona, per default, dentro la coda con priorità piu basssa.
2. Marcare i pacchetti ICMP come 0x00. Vogliamo pingare e mostrare la latenza per i pacchetti ad alta priorità.
3. Marcare tutti i pacchetti destinati alla porta 1024 o inferiore come 0x01. Questo da priorità ai servizi di sistema quali Telnet e SSH. Anche la porta di controllo FTP ricade in questo range comunque i dati di trasferimento FTP stanno su porte alte e rimangono nella banda 0x03.
4. Marcare tutti i pacchetti destinati alla porta 25 (SMTP) come 0x03. Se qualcuno invia e-mail con grandi allegati non vogliamo che affoghi il traffico interattivo.
5. Marcare tutti i pacchetti destinati a game server multiplayer come 0x02. Questo da ai giocatori bassa latenza ma evita loro di impantanare le applicazioni di sistema che la richiedono.

   Marcare ogni pacchetto "piccolo" come 0x02. Pacchetti ACK uscenti generati da download entranti devono essere inviati prontamente per assicurare scaricamenti efficienti. Questo è possibile con l'utilizzo dell'iptables length module.

Ovviamente, tutto ciò può essere aggiustato secondo le vostre esigenze.

3.4. Qualche piccola miglioria...

Ci sono altre due cose che potete fare per migliorare la vostra latenza. Primo, potete settare la Maximum Transmittable Unit (mtu) ad un valore più basso del default di 1500 bytes. Abbassando questo numero si abbasserà anche il tempo medio che dovrete aspettare per inviare un pacchetto prioritario se c'è già un pacchetto a bassa priorità full-sized che viene inviato. Abbassando questo numero decrescerà leggermente anche il vostro throughput perché ogni pacchetto contiene almeno 40 byte di valore di IP e TCP header information.

L'altra cosa che potete fare per migliorare la latenza anche sul vostro traffico a bassa priorità è di diminuire la lunghezza della coda dal valore di default di 100, la quale può impiegare anche 10 secondi per svuotarsi con una mtu di 1500 byte.

3.5. Provare a limitare il traffico in arrivo

Usando l'Intermediate Queuing Device (IMQ), possiamo elaborare tutti i pacchetti entranti attraverso una coda nello stesso modo in cui lo facciamo con gli uscenti. La priorità dei pacchetti è piu semplice in questo caso. Potendo solo (provare a) controllare il traffico TCP in arrivo, potremmo mettere tutto il traffico non TCP nella classe 0x00, e tutto il traffico TCP nella classe 0x01. Potremmo anche mettere i pacchetti TCP "piccoli" nella 0x00 dato che questi sono per la maggior parte pacchetti ACK per dati in uscita che sono gia stati inviati. Creiamo una coda standard FIFO nella classe 0x00, e una coda Random Early Drop (RED) nella classe 0x01. La coda RED è migliore della FIFO (tail-drop) per controllare il TCP perché scarta i pacchetti prima di andare in overflow nel tentativo di rallentare trasferimenti che potrebbero andare fuori controllo. Possiamo limitare entrambe le classi ad un valore massimo inferiore all'effettivo valore in ingresso sul modem ADSL.

3.5.1. Perché limitare il Traffico in arrivo non è così giusto

Vogliamo limitare il nostro traffico in ingresso per evitare di riempire la coda presso l'ISP,il quale può a volte bufferizzare fino a un valore di 5 secondi dei dati. Il problema è che attualmente l'unico modo per limitare il traffico TCP in ingresso è quello di scartare pacchetti perfettamente validi. Questi pacchetti hanno già preso una certa parte di banda sul modem ADSL solo per essere scartati dalla Linux Box nello sforzo di rallentare i futuri pacchetti. Essi saranno eventualmente ritrasmessi consumando ancora più banda. Quando limitiamo il traffico, limitiamo il rate dei pacchetti che saranno accettati nella nostra rete. Poiché l' *attuale* data rate in ingresso è leggermente superiore a causa dei pacchetti che scartiamo, dobbiamo limitare il downstream ad un valore *molto* più basso rispetto all'attuale valore del modem ADSL per assicurare una latenza bassa. In pratica devo limitare il mio downstream di 1.5 Mbit/s a 700kbit/sec per avere una latenza accettabile con 5 download concorrenti. Più sessioni TCP si hanno, più larghezza di banda si spreca con i pacchetti scartati, e più basso sarà il valore massimo da configurare.

Un miglior modo di controllare il traffico in arrivo potrebbe essere la manipolazione della larghezza della finestra del TCP, ma al momento di questa stesura non ci sono (libere) implementazioni di ciò per Linux (per quanto ne so...).

4. Implementazione

Dopo tutte le spiegazioni è giunto il momento di implementare il bandwidth management con Linux.

4.1. Avvertimenti

Limitare il rate dei dati inviati al modem DSL non è semplice come può sembrare. La maggior parte di questi modem sono in realtà solo dei bridge ethernet che mandano i dati avanti e indietro tra la vostra linux box e il gateway dal vostro ISP. Molti modem DSL utilizzano ATM come strato di collegamento per inviare dati. ATM invia dati in celle che hanno una lunghezza fissa di 53 byte. 5 di questi byte sono informazioni di intestazione, e i rimanenti 48 byte sono disponibili per i dati. Anche se mandate 1 byte di dati, vengono consumati 53 byte di larghezza di banda poiché le celle ATM sono sempre lunghe 53 byte. Questo significa che se voi state inviando un tipico pacchetto TCP di ACK esso consiste di 0 byte di dati + 20 byte di header TCP + 20 byte di header IP + 18 byte di header Ethernet . Attualmente, anche se il pacchetto ethernet che inviate ha solo 40 byte di payload (TCP e IP header), il payload minimo per un pacchetto Ethernet è di 46 byte di dati, così i rimanenti 6 byte sono riempiti con dati nulli. Questo significa che l'attuale lunghezza del pacchetto Ethernet più l'intestazione è di 18 + 46 = 64 byte. Per inviare 64 byte su ATM, dovete inviare due celle ATM che consumano 106 byte di larghezza di banda. Quindi per ogni pacchetto TCP di ACK, state sprecando 42 byte. Questo potrebbe essere okay se Linux tenesse conto della incapsulazione che usa il modem DSL, invece, Linux tiene solo conto di TCP header, IP header e dei 14 byte dell'indirizzo MAC (Linux non tiene conto dei 4 byte di CRC poiché questo viene fatto a livello hardware). Linux non tiene conto inoltre né della grandezza minima del pacchetto Ethernet di 46 byte, né della grandezza fissa della cella ATM.

Tutto ciò sta a significare che dovete limitare la vostra larghezza di banda in uscita ad un valore inferiore rispetto all'effettiva capacità (fintanto che si possa calcolare uno schedulatore di pacchetti che possa tener conto dei vari tipi di incapsulazioni che vengono utilizzate). Potreste scoprire di aver calcolato un buon valore a cui limitare la vostra larghezza di banda, ma poi scaricate un grosso file e la latenza inizia a schizzare verso l'alto dopo 3 secondi. Ciò di solito avviene perché la larghezza di banda che questi piccoli pacchetti ACK consumano viene calcolata male da Linux.

Ho lavorato a questo problema per qualche mese e ho quasi aggiustato una soluzione che presto rilascerò al pubblico per ulteriori test. La soluzione implica l'utilizzo di una coda user-space invece della QoS di Linux per limitare il rate dei pacchetti. Ho fondamentalmente implementato una semplice coda HTB usando una coda linux user-space. Questa soluzione (per quanto ne so) riesce a regolare il traffico in uscita MOLTO BENE anche durante un massiccio download (diversi flussi) e pesante upload (gnutella, diversi flussi) la latenza ha un picco di 400ms oltre la mia latenza senza traffico di circa 15ms. Per maggiori informazioni su questo metodo QoS, iscrivetevi alla mailing list per le novità o controllate gli aggiornamenti a questo HOWTO.

4.2. Script: myshaper

Il seguente è il listato dello script che utilizzo per controllare la larghezza di banda sul mio router Linux. Esso utilizza molti dei concetti trattati nel documento. Il traffico in uscita viene messo in una delle 7 code a seconda del tipo. Il traffico in entrata è posizionato in due code con i pacchetti TCP che vengono scartati prima (priorità più bassa) se i dati in entrata vanno oltre il rate fissato. I valori dati in questo script sembrano essere OK per il mio setup ma i vostri risultati potrebbero variare.

> Questo script era basato in origine sul WonderShaper ADSL come visto presso il sito web LARTC (http://www.lartc.org).

```
#!/bin/bash
#
# myshaper - DSL/Cable modem outbound traffic shaper and prioritizer.
#            Based on the ADSL/Cable wondershaper (www.lartc.org)
```

```
#
# Written by Dan Singletary (8/7/02)
#
# NOTE!! - This script assumes your kernel has been patched with the
#          appropriate HTB queue and IMQ patches available here:
#          (subnote: future kernels may not require patching)
#
#       http://luxik.cdi.cz/~devik/qos/htb/
#       http://luxik.cdi.cz/~patrick/imq/
#
# Configuration options for myshaper:
#  DEV    - set to ethX that connects to DSL/Cable Modem
#  RATEUP - set this to slightly lower than your
#           outbound bandwidth on the DSL/Cable Modem.
#           I have a 1500/128 DSL line and setting
#           RATEUP=90 works well for my 128kbps upstream.
#           However, your mileage may vary.
#  RATEDN - set this to slightly lower than your
#           inbound bandwidth on the DSL/Cable Modem.
#
#
#  Theory on using imq to "shape" inbound traffic:
#
#     It's impossible to directly limit the rate of data that will
#  be sent to you by other hosts on the internet.  In order to shape
#  the inbound traffic rate, we have to rely on the congestion avoidance
#  algorithms in TCP.  Because of this, WE CAN ONLY ATTEMPT TO SHAPE
#  INBOUND TRAFFIC ON TCP CONNECTIONS.  This means that any traffic that
#  is not tcp should be placed in the high-prio class, since dropping
#  a non-tcp packet will most likely result in a retransmit which will
#  do nothing but unnecessarily consume bandwidth.
#     We attempt to shape inbound TCP traffic by dropping tcp packets
#  when they overflow the HTB queue which will only pass them on at
#  a certain rate (RATEDN) which is slightly lower than the actual
#  capability of the inbound device.  By dropping TCP packets that
#  are over-rate, we are simulating the same packets getting dropped
#  due to a queue-overflow on our ISP's side.  The advantage of this
#  is that our ISP's queue will never fill because TCP will slow it's
#  transmission rate in response to the dropped packets in the assumption
#  that it has filled the ISP's queue, when in reality it has not.
#     The advantage of using a priority-based queuing discipline is
#  that we can specifically choose NOT to drop certain types of packets
#  that we place in the higher priority buckets (ssh, telnet, etc).  This
#  is because packets will always be dequeued from the lowest priority class
#  with the stipulation that packets will still be dequeued from every
#  class fairly at a minimum rate (in this script, each bucket will deliver
#  at least it's fair share of 1/7 of the bandwidth).
#
#  Reiterating main points:
#   * Dropping a tcp packet on a connection will lead to a slower rate
#     of reception for that connection due to the congestion avoidance algorithm.
#   * We gain nothing from dropping non-TCP packets.  In fact, if they
#     were important they would probably be retransmitted anyways so we want to
```

```
#     try to never drop these packets.  This means that saturated TCP connections
#     will not negatively effect protocols that don't have a built-in retransmit like TCP.
#   * Slowing down incoming TCP connections such that the total inbound rate is less
#     than the true capability of the device (ADSL/Cable Modem) SHOULD result in little
#     to no packets being queued on the ISP's side (DSLAM, cable concentrator, etc).  Since
#     these ISP queues have been observed to queue 4 seconds of data at 1500Kbps or 6 megabits
#     of data, having no packets queued there will mean lower latency.
#
#  Caveats (questions posed before testing):
#   * Will limiting inbound traffic in this fashion result in poor bulk TCP performance?
#     - Preliminary answer is no!  Seems that by prioritizing ACK packets (small <64b)
#       we maximize throughput by not wasting bandwidth on retransmitted packets
#       that we already have.
#

# NOTE: The following configuration works well for my
# setup: 1.5M/128K ADSL via Pacific Bell Internet (SBC Global Services)

DEV=eth0
RATEUP=90
RATEDN=700  # Note that this is significantly lower than the capacity of 1500.
            # Because of this, you may not want to bother limiting inbound traffic
            # until a better implementation such as TCP window manipulation can be used.

#
# End Configuration Options
#

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

# Reset everything to a known state (cleared)
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

###########################################################
#
# Outbound Shaping (limits total bandwidth to RATEUP)

# set queue size to give latency of about 2 seconds on low-prio packets
ip link set dev $DEV qlen 30

# changes mtu on the outbound device.  Lowering the mtu will result
# in lower latency but will also cause slightly lower throughput due
# to IP and TCP protocol overhead.
ip link set dev $DEV mtu 1000

# add HTB root qdisc
tc qdisc add dev $DEV root handle 1: htb default 26

# add main rate limit classes
tc class add dev $DEV parent 1: classid 1:1 htb rate ${RATEUP}kbit

# add leaf classes - We grant each class at LEAST it's "fair share" of bandwidth.
#                    this way no class will ever be starved by another class.  Each
#                    class is also permitted to consume all of the available bandwidth
#                    if no other classes are in use.
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

# attach qdisc to leaf classes - here we at SFQ to each priority class.  SFQ insures that
#                                within each class connections will be treated (almost) fairly.
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

# filter traffic into classes by fwmark - here we direct traffic into priority class according to
#                                         the fwmark set on the packet (we set fwmark with iptables
#                                         later).  Note that above we've set the default priority
```

```
#                                         class to 1:26 so unmarked packets (or packets marked with
#                                         unfamiliar IDs) will be defaulted to the lowest priority
#                                         class.
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

# add MYSHAPER-OUT chain to the mangle table in iptables - this sets up the table we'll use
#                                                      to filter and mark packets.
iptables -t mangle -N MYSHAPER-OUT
iptables -t mangle -I POSTROUTING -o $DEV -j MYSHAPER-OUT

# add fwmark entries to classify different types of traffic - Set fwmark from 20-26 according to
#                                                             desired class. 20 is highest prio.
iptables -t mangle -A MYSHAPER-OUT -p tcp --sport 0:1024 -j MARK --set-mark 23 # Default for low por
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, low
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) - high
iptables -t mangle -A MYSHAPER-OUT -p udp -j MARK --set-mark 21                # DNS name resolution
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 - we don't kn
iptables -t mangle -A MYSHAPER-OUT -p tcp --sport http -j MARK --set-mark 25   # Local web server
iptables -t mangle -A MYSHAPER-OUT -p tcp -m length --length :64 -j MARK --set-mark 21 # small packe
iptables -t mangle -A MYSHAPER-OUT -m mark --mark 0 -j MARK --set-mark 26      # redundant- mark any

# Done with outbound shaping
#
####################################################

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

# uncomment following line if you only want upstream shaping.
# exit

####################################################
#
# Inbound Shaping (limits total bandwidth to RATEDN)

# make sure imq module is loaded

modprobe imq numdevs=1

ip link set imq0 up

# add qdisc - default low-prio class 1:21
```

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

# add main rate limit classes
tc class add dev imq0 parent 1: classid 1:1 htb rate ${RATEDN}kbit

# add leaf classes - TCP traffic in 21, non TCP traffic in 20
#
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

# attach qdisc to leaf classes - here we at SFQ to each priority class.  SFQ insures that
#                                within each class connections will be treated (almost) fairly.
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

# filter traffic into classes by fwmark - here we direct traffic into priority class according to
#                                         the fwmark set on the packet (we set fwmark with iptables
#                                         later).  Note that above we've set the default priority
#                                         class to 1:26 so unmarked packets (or packets marked with
#                                         unfamiliar IDs) will be defaulted to the lowest priority
#                                         class.
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

# add MYSHAPER-IN chain to the mangle table in iptables - this sets up the table we'll use
#                                                         to filter and mark packets.
iptables -t mangle -N MYSHAPER-IN
iptables -t mangle -I PREROUTING -i $DEV -j MYSHAPER-IN

# add fwmark entries to classify different types of traffic - Set fwmark from 20-26 according to
#                                                             desired class. 20 is highest prio.
iptables -t mangle -A MYSHAPER-IN -p ! tcp -j MARK --set-mark 20             # Set non-tcp packets
iptables -t mangle -A MYSHAPER-IN -p tcp -m length --length :64 -j MARK --set-mark 20 # short TCP pa
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              # redundant- m

# finally, instruct these packets to go through the imq0 we set up above
iptables -t mangle -A MYSHAPER-IN -j IMQ

# Done with inbound shaping
#
####################################################

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

5. Test della nuova coda

Il modo più semplice per testare la vostra nuova configurazione è saturare l'upstream con traffico a bassa priorità. Questo dipende da come avete settato le priorità. Per beneficio di esempio, diciamo che avete messo il traffico telnet e il traffico ping alla priorità più alta (fwmark basso) rispetto ad altre porte alte (che sono utilizzate per trasferimenti FTP, ecc.). Se iniziate un upload FTP per saturare la larghezza di banda, dovreste solo notare che il vostro tempo di ping al gateway (dall'altro lato della linea DSL) incrementa di un piccolo valore rispetto all'aumento che si avrebbe senza priorità di coda. Tempi di risposta del ping inferiori a 100ms sono tipici a seconda di come avete configurato il tutto. Tempi di Ping superiori a uno o due secondi indicano probabilmente che le cose non stanno funzionando bene.

6. OK Funziona!! E adesso?

Ora che avete iniziato con successo il managing della vostra larghezza di banda, cominciate a pensare ad un modo per usarla. Dopo tutto, probabilmente la state pagando!

- Usate un client Gnutella e CONDIVIDETE I VOSTRI FILE senza effetti negativi sulla performance della vostra rete
- Fate girare un server web senza che gli "hits" alle pagine vi rallentino a Quake

7. Link correlati

- Bandwidth Controller per Windows - http://www.bandwidthcontroller.com
- dsl_qos_queue (http://www.sonicspike.net/software#dsl_qos_queue) - (beta) per Linux. Nessuna patch al kernel e migliore performance -