Dynamic Rate Limiting in an ISP Setting

David Newall

Tellurian Pty Ltd

ABSTRACT

How can a small ISP offer unlimited, high speed downloads, for a fixed, monthly fee, and avoid going broke? Tellurian operates a small ISP, *rebel.net.au*, and the answer, at least for us, is that it can't—but we can come acceptably close.

Having commissioned a new, high speed, pay-per-byte link, we naturally want our subscribers to enjoy its full benefit, but we also need to control our costs. The solution that we implemented counts bytes downloaded by each subscriber, and rate limits their connection when they reach a download cap. Our subscribers can continue to use the Internet, even after they have reached the cap, but data arrives much more slowly. The limited service means they can continue to work, but gives them a strong incentive to finish quickly and log off.

A monitor is notified, by PPP, at the start and end of each session. It records the number of bytes downloaded at five minute intervals. When a subscriber reaches the download cap (or if they are already at that level at the start of their session) the monitor installs a rate limiting rule into our router. The five-minute usage records are kept for seven days, thus a large download today will count toward the cap for a complete week. This removes any benefit in 'hoarding' downloads until the end of the week, and helps smooth network use.

Subscribers view their current usage records via a web page. The number of bytes downloaded during the previous seven days is displayed in a cumulative graph. Subscribers use this to plan their Internet use.

The system has proven successful. Most users never even noticing it, yet consumption has dropped by 30%, and almost no adverse comments have been received.

XXIII, August, MMI

Dynamic Rate Limiting in an ISP Setting

David Newall

Tellurian Pty Ltd

Introduction

How can a small ISP offer unlimited, high speed downloads, for a fixed, monthly fee, and avoid going broke? This is a vexing issue for many ISPs. Subscribers want unrestricted modem access with no engaged signal; high speed transfers with no download quota; and all for a small, fixed, monthly fee. As the operator of *rebel.net.au*, we would like to be able to meet these needs, but we cannot. We can, however, come acceptably close.

In this paper I focus on subscribers' desire for unlimited quantities of data. In particular, I describe how we introduce a 'speed limit' on subscribers when they reach a download cap. The speed limit makes Internet use painfully slow, but is sufficient to continue working. From the perspective of the subscriber, slow is infinitely better than zero, and much preferred to paying punitive excess-download fees.

History

Our service started with a 'permanent' 64Kbps ISDN link, for which we paid a fixed, monthly fee, regardless of how much data we brought in. This permitted us to totally ignore the quantity of data consumed by each of our subscribers. We later realised that it was a mistake to be so unconcerned by our subscribers' download volumes. While we had few modems—we started with just two—it was barely possible to saturate the link, and performance was very good. As our subscriber-base grew, so too did the size of our modem pool. When we had six modems, performance was sometimes bad, but mostly okay. By the time we grew to ten modems, performance was quite terrible most of the time.

In December, 2000, we decided to upgrade our Internet link. 256Kbps was the minimum speed we felt we could live with, and more would be better. We eventually ordered a 2Mbps fibre service, with data charged per Mbyte downloaded. Even if consumption remained the same, our total data cost would double, but we were certain that the 32-fold increase in bandwidth would result in a significant increase in consumption. We estimated that our costs would triple unless we took steps to control usage.

Solution

We developed a dynamic rate limiting system. It is comprised of a rate limiter and a usage monitor, plus some utilities and a web page.

The Rate Limiter

Our router uses FreeBSD, which includes an IP firewall and traffic shaper, *ipfw*, as standard. An *ipfw* configuration is made up of a list of numbered rules, which is scanned for each packet until a match is found. As we also use our border router as a firewall, we divide the rule number-range into three sections, with the rate limit rules appearing before the firewall rules. All packets matched by rate limit rules are subsequently processed by the firewall rules.

We wrote a program, *limit*, to install, list and remove rate limit rules in the router. It is started by *inetd*, and supports a limited vocabulary: HELP, LOGIN, BYE, and ADD, REMOVE, CLEAR and LIST. The HELP command displays a brief help message. The LOGIN command is used to authenticate the remote user. The BYE command terminates the session, leaving all rate limits intact. The ADD and REMOVE commands insert or remove limits for specified hosts. Requests to add an existing limit, or to remove a non-existent limit, are silently ignored. The CLEAR command removes all rate limits. The LIST command displays the rate-limited addresses.

Rate limit rules are simple. They match packets destined for the target address, that are received on the external network interface. Matching packets are passed through a rate limiting pipe, there being a separate pipe for each rule. The rate limiter assigns and configures the pipes, setting them to 3Kbps bandwidth and 10KByte size. This permits approximately 300 bytes per second transfer rate, which is unashamedly slow: the goal is, after all, to limit subscribers' usage.

The Monitor

The usage monitor, *byteshare*, measures and records traffic volumes on all PPP devices, and sends commands to *limit* to install and remove rate limits as usage exceeds and falls below the cap. *Byteshare* reads a usage file to determine past usage for all subscribers, and adds to the total (and to the file) as they download data.

At start-up, it constructs an in-memory usage table, which contains each users quota and the usage recorded to date. This table is constructed by reading */etc/passwd*, a quota file, and a usage file, and then scans the */var/run* directory to determine which PPP units are in use, and by whom. Finally, it interrogates each active PPP unit to determine the number of bytes already transferred during current sessions.

Byteshare maintains a named socket, which PPP uses to signal start or end of sessions. Whenever a session starts or ends, the PPP *ip-up* and *ip-down* scripts write the PPP unit number, login name and IP address to that socket, as well as to a file in */var/run*. *Byteshare* needs this information to associate the correct subscriber with their current IP address and PPP unit.

Ever five minutes, *byteshare* interrogates all active PPP units, updating both the in-memory usage table and the on-disk usage file. Each record in the usage file stores the subscribers' UID, the system clock and the number of bytes transferred in and out since the last measurement. When a subscriber exceeds his or her quota a rate limit rule is inserted in the router. Rate limits are also inserted or removed, as required, at session start time.

Although *byteshare* records inbound traffic separately from outbound, we only use inbound for rate limit purposes.

Security

The rate limiter requires a one-time challenge-response style login to be completed before any of the rate management commands may be used. A challenge is generated based on the process ID, the system clock, and a sequence number which is incremented with every authentication attempt. This challenge is presented to the caller, who must use a shared, private secret to calculate the correct response. This response is returned in the LOGIN command. If the response is wrong a new challenge is presented.

A 64 bit message authentication code (MAC) is produced from the challenge, using DES in cipher block chaining (CBC) mode. This MAC is formatted as an eleven character string by mapping each consecutive six bits onto the 64 character alphabet comprised of dot, slash, the ten digits, the upper case letters and the lower case letters. The last character is produced by padding the MAC with two zero-bits.

The first eight bytes of the secret are used for the DES key and the second are used for the initialisation vector. If the secret is shorter than 16 bytes long it is padded with NULs.

The DES CBC MAC code was derived from the BSD *bdes* [1,2] utility.

Utilities

We wrote three utilities to assist in management of the usage file. *Byteconvert* converts a usage file from binary to display format, or vice versa. In the style of *fixwtmp*, this utility can be used to patch a file should it become faulty.

Byteusage summarises a usage file, displaying the total number of bytes transferred for each user, or for a specified subset of users, and optionally displays each individual usage record.

Finally, *prunelog* splits a usage file into two separate files, one containing records older than a nominated age, and the other containing records younger than that age. After splitting the usage file, the original file is removed, the file containing the most recent records is renamed to replace it, and *byteshare* is restarted. This is the process by which usage records are retired from quota caluclations. We prune the

```
$ byteconvert -n < /var/local/bytes
 544 997972561 343788  86631
 602 997972561    605    435
 …
 543 997972561   5382   2058
$ echo "199 Fri Aug 17 00:07:06 2001 123 456" | \
     byteconvert -b >> /var/local/bytes
```

Sample use of *byteconvert*

```
$ byteusage | sort +1n
USER                IN       OUT
flamencoareti      9846      8173
cpjpm             21822     22467
john              59462     16210
…
crowther       41890405   4715042
susannes       56843875   2977139

$ byteusage -v davidn
USER                    TIME              IN    OUT
davidn       Sat Aug 18 18:03:06 2001     52      0
davidn       Sat Aug 18 18:07:33 2001 120505   7555
…
davidn       Thu Aug 23 07:52:55 2001  40959   3050
USER                IN       OUT
davidn          1887318    317019
```

Sample output from *byteusage*

usage file three times per day, keeping only the most recent seven days of data each time.

Rather than require old records to be removed from the usage file so frequently, it would be better if *byteshare* maintained a second file descriptor pointing to the oldest, current record, and read and subtracted records from current usage as they reached seven days. There is no reason not to do it this way, and a future version will incorporate this improvement.

Web Page

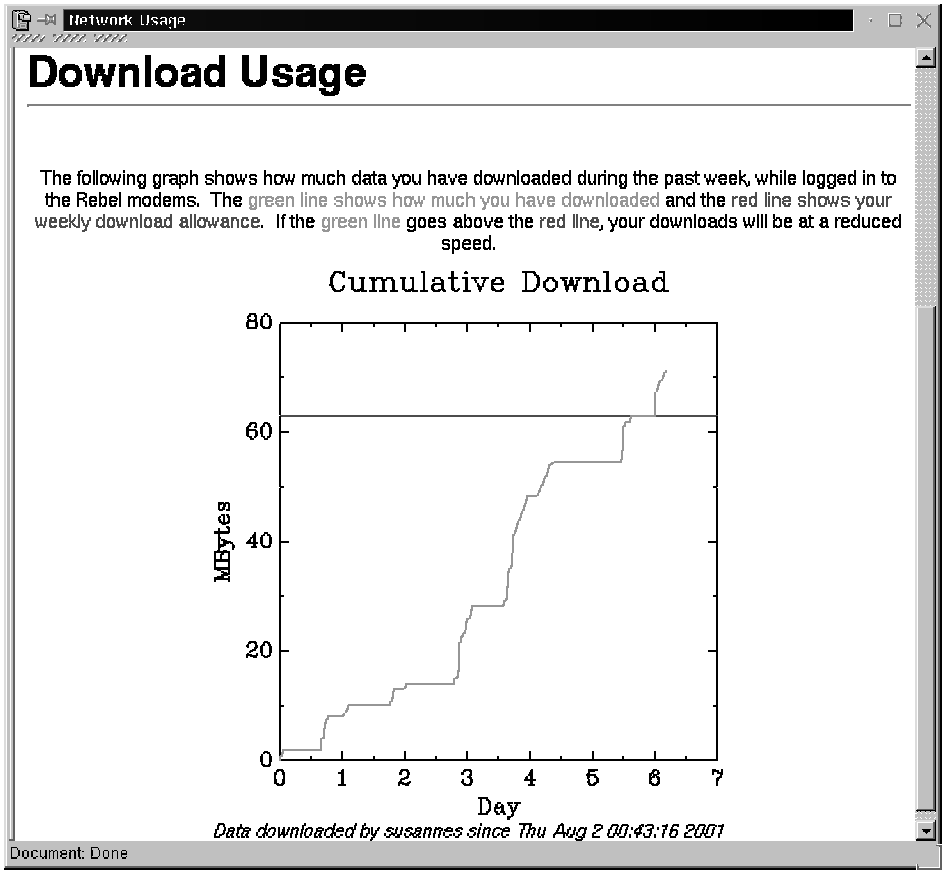
Although it is obvious to subscribers when they have been rate limited, we never the less provide a CGI web page to permit them to view their current recorded usage. This is a single, very simple, Bourne shell script. It determines the subscriber's login name using *who*, retrieves their current usage records using *byteconvert*, and processes the data using a one line *awk* program, which produces cumulative values that are graphed using GNU's *plotutils* Data is never more than five minutes out of date, that being the period of time between updates of the usage file.

Subscriber's report that this page is useful for planning their Internet use.

Tuning

The purpose of this system is to avoid going broke, and it is important to make sure that tunable parameters are set appropriately. The two major parameters are the download cap, which determines the dollar value of high speed data, and the rate-limited bandwidth, which determines the dollar value of excess data. At prevailing wholesale rates of 10 to 20 cents per megabyte, excess data can cost a maximum of

$$\frac{30\ days \times 86400\ seconds \times 300\ bytes}{1000000} \times \$0.20 = \$155.40$$



Web page showing subscriber's current usage

per subscriber per month. With typical monthly fees being $20 to $30, subscribers could send an ISP broke just on excess data, however performance when rate limited really is so slow that this does not seem to be a problem. Should this become a problem, a simple solution would be to modify *limit* so that all rate limit rules pass packets through a single pipe, instead of through one pipe per IP address, thus the cost would be a maximum of $155.40 per month for all subscribers.

We give a monthly quota of 150Mbytes. The next issue is whether to give this all at once, or to dole it out in smaller quantities. Is it better to permit subscribers to use their download allowance in one day, and for them to have lousy speed for the next 30 days, or to give them a smaller download allowance and a correspondingly smaller time before that allowance is renewed? This is a user-satisfaction issue, and is the sort of question which elicits many different answers. We choose to split the monthly quota into four weekly quotas of 63Mbytes each, which seems a good balance. It keeps the time before quota is regenerated fairly short, yet provides an amount of data sufficient to download quite large files.

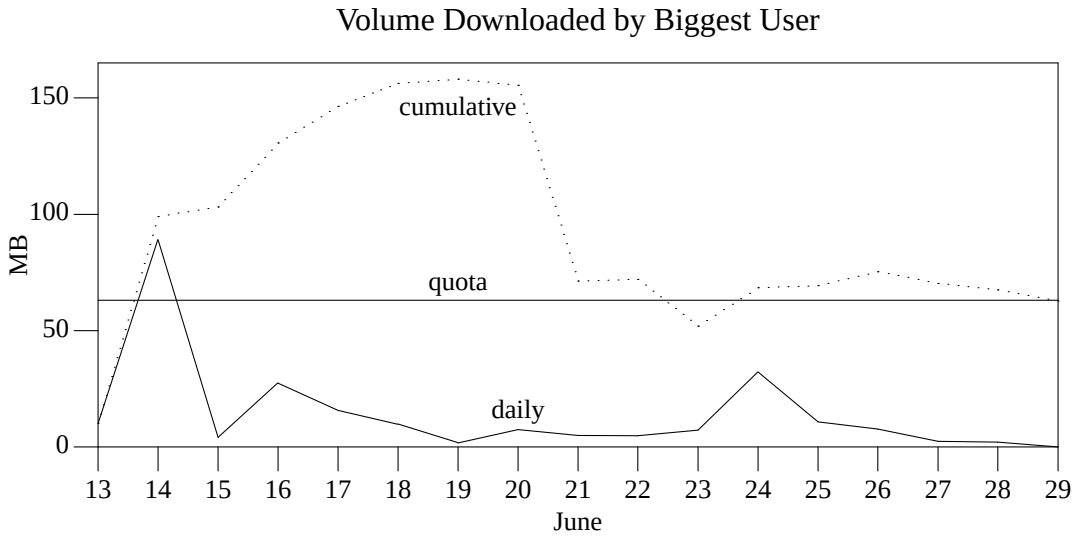
Results

The system has proven very successful. We have recorded a dramatic drop in consumption, starting when we introduced rate limiting. Prior to rate limiting we were bringing in 300 to 400Mbytes of data per day. In mid June the numbers drop to 200 to 250Mbytes per day, and continued to decline the current volume of 100 to 200Mbytes per day.

Such a dramatic decrease called for investigation. Had we completely misjudged what effect rate limiting would have on our subscribers? In fact we had not. We discovered that one subscriber stayed online for about twenty hours every day, and this had been the case for a considerable period of time. This was within the bounds of our 'fair share' policy, and we would not have been concerned even if we had

known about it. He was also transferring data, more or less constantly, which equated to about 180 Mbytes every day. With dynamic rate limiting, he was restricted to about 20 Mbytes per day.

We pre-announced the rate limiter to ensure that our subscribers would know what to expect. This particular user did indeed notice the effect fairly quickly. He logged off for a few hours on day three. (Notice the dip at June 15 in the graph, below.) On the evening June 18 he logged off again and remained off all night. The next morning, at 7.45am, he logged on again, saw that he was still rate limited, and logged off again. He was online for a mere two minutes. We considered that to be a success!



References

[1] Source code is distributed with FreeBSD in the directory, `/usr/src/secure/usr.bin/bdes`.

[2] Matt Bishop, *Implementation Notes on bdes(1)*, Technical Report PCS-TR-91-158, Department of Mathematics and Computer Science, Dartmouth College, Hanover, NH 03755 (Apr. 1991).