Short Evaluation of Linux's Token-Bucket-Filter (TBF) Queueing Discipline

Kurt Wagner
kwagner@cosy.sbg.ac.at

17th May 2001

1 Introduction

A general overview of most of the terms used in this document can be found in RFC 2475 [1]. Additional information about traffic control in Linux can be found in [3] and [4]. To install and manipulate individual traffic control elements the *tc* user-space application is used [2].

2 Token Bucket

In the token bucket metaphor, tokens are generated at a certain rate; the bucket has a specified capacity called size or burst size. If the bucket fills to capacity, newly arriving tokens are discarded. Each token represents permission to send a certain number of bits into the network. To transmit a packet, a number of tokens corresponding to the packet size must be remove from the bucket .

If there are not enough tokens in the bucket, the packets exceed the particular rate limit given by the token bucket. If the bucket is already full, incoming tokens overflow and are not available for future packets. Thus, at any time the largest burst a source can send into the network is roughly proportional to the size of the bucket [5].

A token bucket meter is used to manage a device that regulates the traffic. The token bucket itself has no discard or priority police. Rather the token bucket discards tokens and leaves to the policer/shaper the problem of what to do with conforming/exceeding packets. In the test scenarios presented in this paper, conforming packets are simply sent into the network, non-conforming packets are dropped (policer) or delayed (shaper), but generally different policies may be used (e.g. marking conforming/exceeding packets).

A token bucket mechanism used for traffic shaping has both a token bucket and a data buffer, or queue; without the data buffer it would simply be a policer. Arriving packets that cannot be sent immediately are delayed in the data buffer, thus shaping the traffic.

A token bucket permits burstiness within bounds. It guarantees that the burstiness is bounded so that the source will never send much faster than the token bucket's capacity. It also guarantees that the long-term transmission rate will not exceed the established rate at which tokens are placed in the bucket.

2.1 Linux Token Bucket Filter (TBF)

As shown in Figure 1 there are three basic parameters for each TBF queueing discipline. *Rate* is the rate the bucket refills with tokens - which represents the average transmission rate of a traffic flow. The *bucket size* or *burst size* is the number of tokens the token bucket can

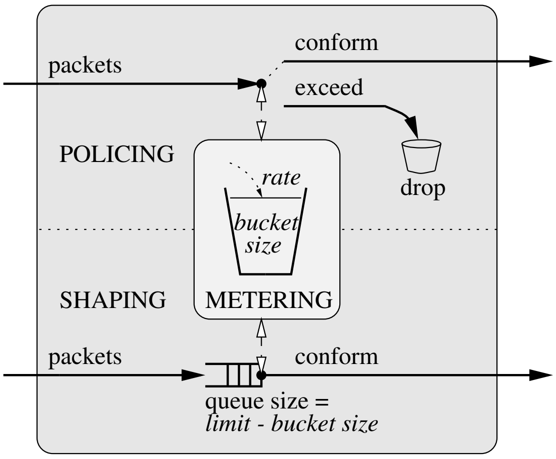

Figure 1: Linux TBF

store. Linux uses a byte-count token bucket, so every token is equivalent to one byte. The *limit* parameter is the sum of the bucket size and the size of the queue; it decides if the TBF shapes or polices. If limit is equal to the bucket size the queue size is zero and non conforming packets are dropped, thus the stream is policed. If limit is greater than the bucket size some non conforming packets are queued (the stream is shaped). So in terms of RFC 2475 [1] the Linux TBF queueing discipline is meter, shaper and policer (dropper) all in one. It is based on a token bucket technique to do metering and according to the value of the limit parameter it shapes or polices the traffic.

Consider a simple example:

```
# tc qdisc add dev eth1 \
  handle 1:0 root tbf \
  rate 1Mbit burst 20Kb limit 20Kb
```

The first line *adds* a queueing discipline to device *eth1*. In the second line we assign the handle *1:0* to the qdisc, state the parent of the qdisc to be the *root* of the device and name *tbf* to be the qdisc we want to add. More information on *tc*'s features and parameters can be found in [6] and [7].

The interesting part of the command is shown in line three. The average rate which can be achieved is 1Mbit/sec, bursts up to 20Kbytes are allowed (=bucket size) and the limit parameter states that the size of the queue is zero. *tc* can print out diagnostic information about the installed queueing disciplines.

```
# tc qdisc show dev eth1
  qdisc tbf 1:  rate 1Mbit
  burst 20Kb lat 0us
```

lat 0us states that a packet can be delayed for a maximum of zero microseconds. Obviously there is a relation between limit and latency. In case of policing (limit = bucket size) there is no latency, in case TBF shapes, maximum delay is estimated as $latency = (limit - burst)/rate$.

2.2 Timing Issues

When TBF throttles it starts a watchdog timer, which determines when it is ready to transmit again. The minimal resolution for this watchdog timer is derived from the timer interrupts. The "timer-interrupt-frequency" is called *system timer* (ST) and it usually ticks at 100Hz in Linux (mainly because the early Linux development was based on slow hardware). Hence, the minimal time-interval to which the watchdog timer can be set is $\frac{1}{ST}$ seconds. If no new packets arrive during this period, or if the device is not woken up by EOI (end of interrupt) for some previous packet, TBF can stop its activity for $\frac{1}{ST}$ seconds.

This means, that with given burst size and timer-interrupt-frequency, the maximal possible rate is $rate_{crit} = burst * ST$. For example, for a 100Mbit/sec Ethernet and ST=100Hz the allowed minimum burst size is 128Kbyte.

To increase the maximum possible rate/burst size it requires speeding up the Linux system timer, e.g. to 1024Hz, the default timer for the Alpha processor[1]. It is

[1]The value for the system timer can be found

also possible on some platforms to choose a more accurate clock[2] [8].

3 Some Tests

3.1 Policing

Figure 2: Test Setup

The setup for the TBF measurements is shown in figure 2. A file is copied from host *source* to *target* with TCP via *gateway.* The token bucket filter is attached to *gateway*s egress interface and the packets leaving the interface are logged via `tcpdump [9]`.

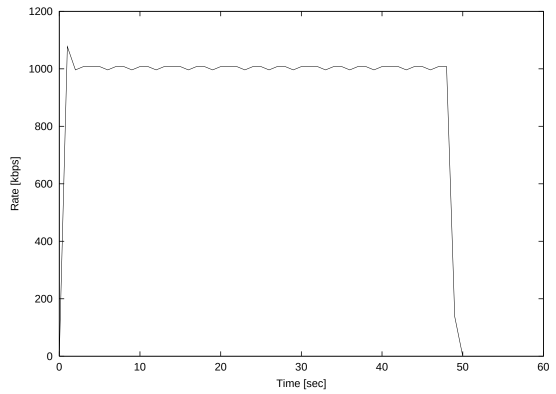

Figure 3: Policing - rate 1Mbit/sec, burst size 20Kbytes

The simple example from above, policing to 1Mbit/sec with 20Kbytes burst size, is visualized in figure 3. As expected the average transmission rate of 1Mbit/sec is achieved quite exactly and the peak at the beginning visualizes the initial burst.

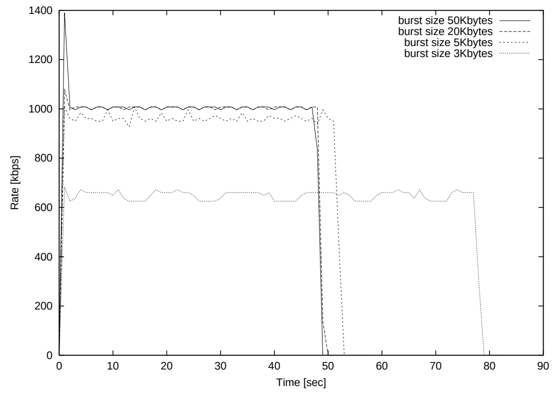

Figure 4: Policing - rate 1Mbit/sec, different burst sizes

Figure 4 shows the same transmission for different burst sizes. The graphs for the transmissions policed with 50Kbytes and 20Kbytes burst size take full advantage of the rate, but choosing smaller burst sizes (somewhere below 5 times MTU in this specific case) causes problems. The theoretically rate of 1Mbit/sec can not be reached anymore. What looks like a TBF issue first, turns out to be standard TCP behavior for congestion control. TCP uses strategies that reduce retransmission when packet delay or loss occurs [10]. Due to this algorithms it is very hard to police a TCP transmission exactly to a specific rate when a small token bucket size is used.

Figure 5 makes this observation even more clear. A TCP and a UDP transmission are compared and the result is visualized in one picture. It is clear that the UDP packets dropped by the TBF are really lost, whereas the dropped TCP packets are retransmitted. Figure 5 shows that it is possible to police to a (more or less) accurate rate even with a small burst size.

3.2 Shaping

An example for traffic shaping is given in figure 6. The limit parameter for the shaped trans-

in `/usr/src/linux/include/asm-i386/param.h` (it is called HZ in Linux).

[2]Take a look at `/usr/src/linux/include/net/-pkt_sched.h` and the comments.

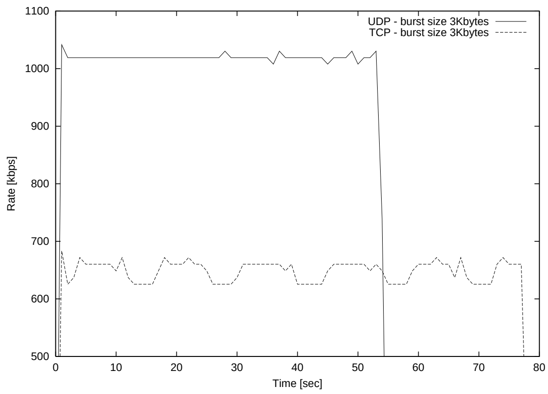

Figure 5: Policing, UDP/TCP comparison - rate 1Mbit/sec

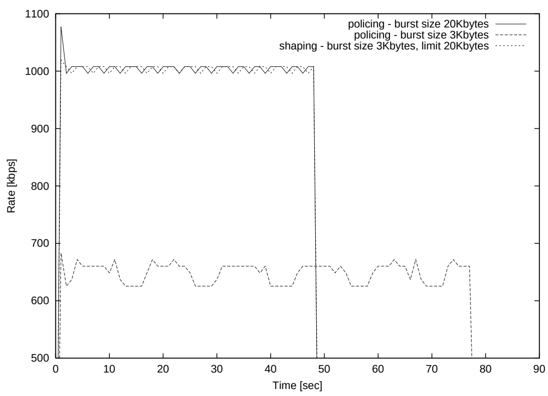

Figure 6: Shaping/policing comparison - rate 1Mbit/sec

mission is greater than the burst size, which means packets that do not obey the token bucket meter are queued, so the traffic stream is shaped. Moreover it can be seen that the shaped (TCP) traffic stream takes full advantage of the available rate, whereas the stream policed with exactly the same rate and burst parameter is not able to do so.

Figure 6 does not reveal anything about the latency the shaped traffic stream experiences. A look at the diagnostic information reveals:

```
# tc qdisc show dev eth1
  qdisc tbf 1:  rate 1Mbit
  burst 3Kb lat 101.8ms
```

101.8 Milliseconds is quite large. estimating the maximal latency with the formula given in section 2.1 gives:

$$Lat = \frac{limit - burst}{rate} = \frac{20-3}{1024/8} = 132.8ms$$

The values are not exactly identical, because *tc* uses a somewhat different formula to calculate the latency[3] . The point here is that one should be aware of the resulting delay and that traffic shaping in software usually requires a large number of additional timer interrupts.

3.3 Double TB - Peakrate

There is one problem in the way a token bucket propagates bursts - it is not possible to guarantee a specific rate that will not be exceeded (called the *peakrate*). That is, because the burst size states how many packets (or how many bytes) can flush through, but it says nothing about the time needed to do so. Consequently, if a token bucket contains tokens equivalent to 100Kbytes and traffic is emitted by the source with 10Mbit/sec, the full 10Mbit/sec are propagated for about 78ms and after that the token bucket throttles to the configured rate. With the tools shown so far

[3] Details can be found at `iproute-`*`version`*`/tc/-q_tbf.c`

there is only one way to guarantee that a traffic stream is really bounded to a specific upper-limit rate: setting the burst size to a very small value, or to be more precise, setting burst size to be equal to the maximum transfer unit (MTU)[4]. With a MTU-sized token bucket the average rate is chosen as upper-bound, but bursts are not propagated anymore.

The TBF queueing discipline solves this problem by metering the data stream with two token buckets. One of them controls steady state burst size, as we have already investigated, another one uses a MTU-sized token bucket to limit bursts to an upper-bound rate at a smaller time scale. If a packet does not conform to both token buckets it is dropped or queued (given the queue size is not zero). The syntax to do this is as follows.

```
# tc qdisc add dev eth1 \
  handle 1:0 root tbf \
  rate 1Mbit burst 200Kb \
  peakrate 1.1Mbit mtu 1520b \
  limit 1520b
```

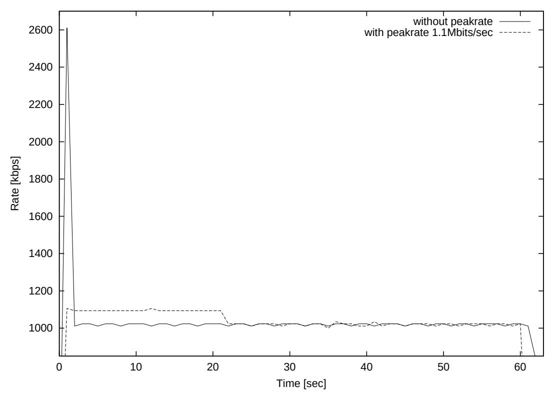

Figure 7: Policing with peakrate - rate 1Mbit/s, burst size 200Kbytes

The limit parameter relates to the smaller, MTU-sized token bucket, so the queue size in this specific case is zero (traffic is policed).

[4]The MTU is the maximum packet size, the biggest packet that will conform to the traffic specifications.

Even though the traffic has to obey two token buckets now, there is still only one queue! But the formula to calculate the maximum latency due to queueing, has to be extended. The maximum possible latency is estimated as:

$$\max(\frac{limit - burst}{rate}, \frac{limit - MTU}{rate})$$

In figure 4 we have already seen the problem associated with policing TCP streams with small burst sizes, so we use this TBF to police UDP traffic. The result can be seen in figure 7 and is as expected.

One last remark to the timing issues mentioned in section 2.2. When the peak rate parameter is used the traffic stream is measured against a very small, MTU-sized token bucket. As $rate_{crit} = burst * ST$ and MTU (=burst) is about 1.5Kbytes in case of Ethernet, the highest possible peak rate with a 100Hz system timer is about 150Kbytes/sec; even with a system timer cranked up to 1KHz it is only about 1.5Mbit/sec! Choosing higher peak rates can lead to unexpected results.

References

[1] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, *An Architecture for Differentiated Services,* RFC 2475, December 1998.

[2] Alexey Kuznetsov, *iproute2 package,* `ftp://ftp.inr.ac.ru/ip-routing/`

[3] Werner Almesberger, *Linux Network Traffic Control - Implementation Overview,* `ftp://lrcftp.epfl.ch/pub/people/almesber/pub/tcio-current.ps.gz`, Technical Report SSC/1998/037, EPFL, November 1998.

[4] Saravanan Radhakrishnan, *Linux - Advanced Networking Overview,* `http://`

qos.ittc.ukans.edu/howto.ps, August 1999.

[5] Andrew Tanenbaum, *Computer Networks*, pp. 379-384, Prentice Hall 1996.

[6] Martijn van Oosterhout, Linux Packet Shaping HOWTO, http://cupid.sunintern.com/~kleptog/Packet-Shaping-HOWTO.html, March 2000.results

[7] Bert Hubert, Gregory Maxwell, Remco van Mook, Martijn van Oosterhout, Paul Schroeder, *Linux 2.4 Advanced Routing HOWTO*, http://www.ds9a.nl/2.4Routing/#download, Oktober 2000.

[8] Moshe Bar. *Keeping the Time.* http://www.byte.com/column/servinglinux/BYT20000202S0006/. February 2000.

[9] The ***tcpdump*** program, http://www.tcpdump.org.

[10] Douglas E. Comer, David L.Stevens, *Internetworking with TCP/IP - Volume II*, pp. 273-277, Prentice Hall 1991.