Traffic-Shaping with fwbuilder under Linux

Dr. Ralf Schlatterbeck
Open Source Consulting

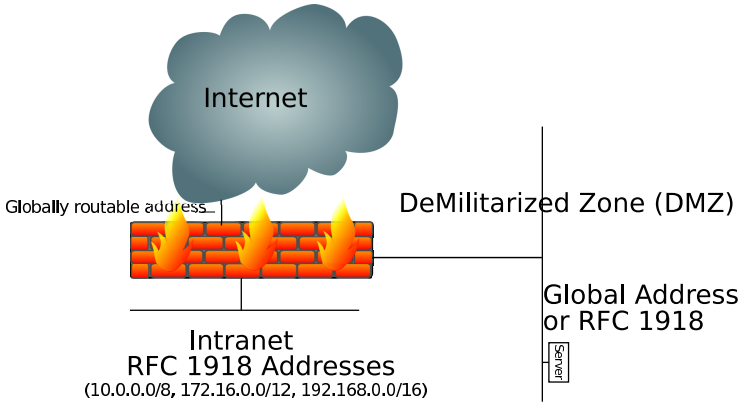
Email: office@runtux.com
Web: http://www.runtux.com
Tel. +43/650/621 40 17

Contents

Initial Situation – Wishes

- Firewall with Intranet and ≥ 1 DMZ segment
- Homegrown `iptables` scripts to be replaced with `fwbuilder` configuration
- Wanted to include traffic shaping in the `fwbuilder` configuration
- `fwbuilder` supports multiple firewalls in a single configuration
- these firewall can share network objects
- nice for several company firewalls at several locations interconnected by (Open-) VPN

Network Diagram



Acknowledgements

These are some things I found on the net – besides the usual tutorials that helped me a lot

- Traffic Shaping with fwbuilder [Sch09] on classification of packets with fwbuilder and shaping with `tc` – as it turned out this doesn't work for inbound shaping
- OpenWRT's `qos-scripts` by *Felix Fietkau* aka `nbd` also use HFSC, RED and SFQ for shaping
- it already contains reclassify targets and inbound and outbound shaping – but uses the `imq` device which is not in the kernel

Acknowledgements

- The /usr/lib/qos/tcrules.awk script from OpenWRT generates `tc` commands and contains a reference to [CJOS01]
- A patch to better document `tc hfsc` – apparently never merged
- Shorewall has special `tc` rules to configure `ifb`
- A blog entry on getting more documentation out of `tc` by issuing clever help commands
- lkml thread that started out modifying the `dummy` device for `imq` functionality and turned into `ifb`

Traffic Control

Mechanisms to Achieve Traffic Control

- Traffic Shaping is often used synonymous to Traffic Control
- decide *which* packets *to accept* at *which rate*
- determine at *which rate* to send packets
- determine in *which order* to send packets
- tremendous power to re-arrange traffic flows
- . . . is no substitute for adequate bandwidth [Bro06]

\+ more predictable usage and bandwidth allocation

– complexity

Traffic Control

- Shaping: delay and/or drop packets to meet the desired rate
- Policing: limit traffic in a particular queue
- Classification: put packets in different classes which are treated differently
- We use the Hierarchical Fair Service Curve (HFSC) Algorithm [SZN00] for classification, shaping and policing
- At the leaf-nodes of the hierarchy we use Random Early Detection (RED) for bulk and Stochastic Fair Queuing (SFQ) for non-bulk traffic

Policy Rules: Traffic Shaping

- traffic classes: Express, Interactive, VPN, Normal, Bulk with corresponding “Mark” actions
- Mark actions defined as TagServices
- Express for time-critical services, (NTP, VoIP, ?Ping)
- Interactive for e.g. SSH
- VPN for site-to-site VPN traffic
- Normal: Web surfing
- Bulk: Downloads
- we can mark packets with such a mark and it will be put into the given traffic class
- everything not marked is Bulk

Example Traffic Shaping Configuration

	Source	Destination	Service	Interface	Direction	Action
	Traffic Shaping (9 rules)					
2	Any	Any	ntp from ntp	All		Express
3	Any	Any	ssh from ssh domain from domain	All		Interactive
4	Any	Any	non-bulk ACK < 128 non-bulk SYN < 128	All		Interactive
5	Any	Any	OpenVPN from OpenVPN OpenVPN from OpenVPN	All		VPN
6	Any	Any	http https from http from https	All		Normal
7	Any	Any	Express > 400	All		Normal
8	Any	Any	Interactive > 1000	All		Normal
9	Any	Any	VPN < 400	All		Interactive

Re-Classification

- Sometimes we want to re-classify some packets depending on their traffic-shaping class
- TCP ACK for non-bulk packet should be faster
- Don’t want too large Express packets
- Don’t want too large Interactive packets, e.g. we want SSH in Interactive but we *don’t* want SCP in Interactive $\rightarrow$ reclassify large SSH packets
- Promote small VPN packets to Interactive
- Many of these need a match on the size of the packet
- we need to define a custom rule in `fwbuilder`

Traffic Control: HFSC

- with HFSC we can form a hierarchy of classes and specify bandwith and realtime requirements
- realtime allocate a slightly higher priority to *new* flows to minimize delay
- . . . even if allocation for other classes is slightly violated (not their realtime guarantees)
- for each class in the hierarchy specify bandwidth
- leaf-nodes specify packet or frame size and delay
- borrow from siblings if these need less bandwidth
- “Fair”: even after borrowing the guaranteed bandwidth isn’t violated

Traffic Control: RED

- Random Early Detection (or “Drop”) [FJ93]
- Even if not yet congested do early notification
- can either (randomly) drop packets early
- . . . or use explicit congestion notification (ECN) TCP option
- observation: with full queues packets of *all* flows are dropped
- . . . which leads to lots of retransmissions
- with RED we get better utilisation
- but it’s hard to configure right [CJOS01, Flo97]

Traffic Control: SFQ

- Stochastic Fair Queuing based on Fair Queuing by John Nagle [Nag85, Nag87]
- large number of queues served round robin (e.g. 128 queues)
- flows are allocated to queues by a hash function
- hash function changes periodically
- end result is a stochastically fair allocation
- . . . which can break with misbehaved nodes (e.g. file-sharing with lots of connections)

⇒ use RED for file-sharing traffic

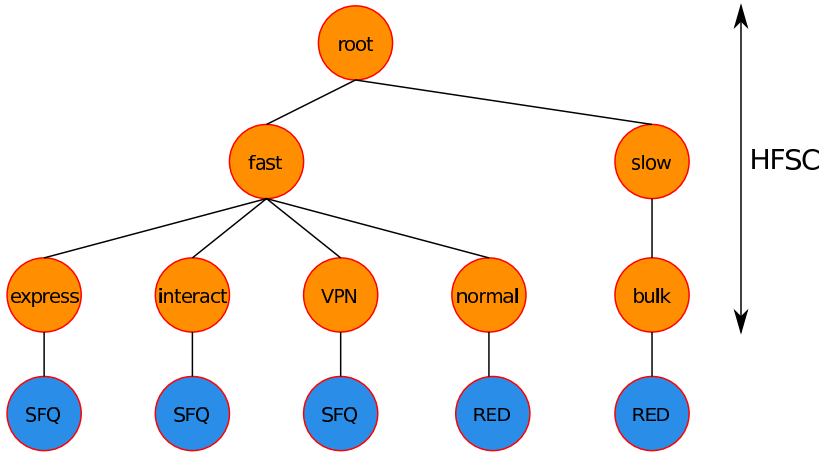
Traffic Control: fwbuilder Configuration

- to generate traffic control configuration we use a hook in fwbuilder called an *epilog script*
- set up special devices for traffic shaping
- generate HFSC, RED, SFQ queueing disciplines (qdiscs)
- work special magic for inbound shaping
- restore OpenVPN-generated routes
- use advanced routing mechanisms where fwbuilder isn’t expressive enough
- in fwbuilder click on the firewall (e.g. “vienna”)
- . . . then on “Firewall Settings . . . ”

Traffic Control: fwbuilder Configuration

```
import sys
from rsclib.trafficshape import Traffic_Class as TC
from rsclib.trafficshape import Shaper
root = TC (100)
fast = TC (90, parent = root)
slow = TC (10, parent = root)
express = TC (1, 128, delay_ms = 10,
   parent = fast, is_bulk = False,
   fwmark = '0x10/0x1f0')
...
shaper = Shaper ("$TC", root)
for interface, bandwidth in \
  (('eth2', 16000 * 0.98), ('ifb0=eth2', 16000 * 0.98)) :
    print shaper.generate (bandwidth, interface)
```

Traffic Control: fwbuilder Configuration



Limits of Traffic Shaping

- we can't really control our incoming traffic
- no chance against malicious traffic, e.g., Distributed Denial of Service (DDoS)
- but TCP has flow control and congestion control mechanisms
- when it doesn't receive an ACK for a packet it will reduce the sending rate
- so we *can* shape traffic by dropping packets
- we need to limit bandwidth to slightly below the maximum so the queue is in our firewall not in the upstream router

Limits of Traffic Shaping

- outbound shaping also needs to reduce the bandwidth
- otherwise we would send at Ethernet speed
- ... and have a long queue in the router
- because the upstream bandwidth is below Ethernet speed

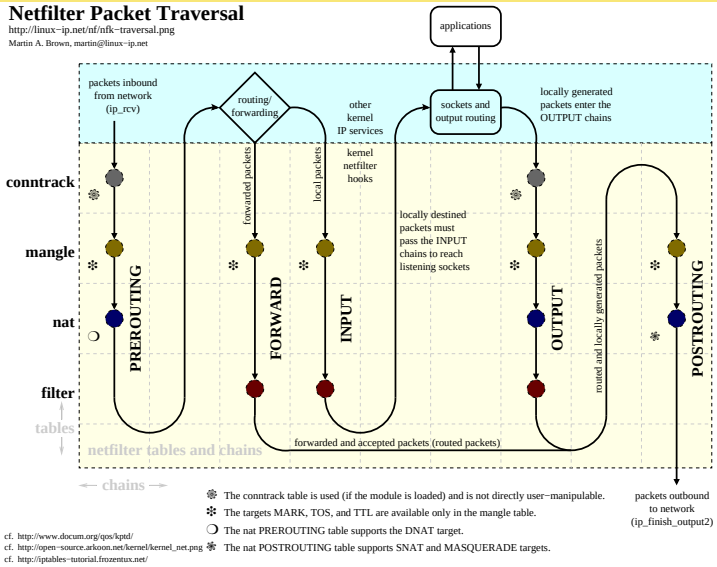
→ use same mechanism for inbound and outbound shaping

- works quite well if inbound traffic is mainly TCP

Linux and Inbound Shaping

- Linux can't do inbound shaping
- trick: redirect traffic to Intermediate Functional Block (`ifb`) device
- ... and there we do *outbound* shaping
- after shaped traffic leaves `ifb` it is reinserted at the point where we redirected
- unfortunately redirect happens *before* the `mangle` table in the PREROUTING chain
- so we need a different mechanism to classify traffic
- remember: for outbound traffic we use firewall marks to classify traffic for shaping

Linux-Kernel Packet Travel



Linux and Inbound Shaping

- we don't want different mechanisms for classifying inbound and outbound traffic
- solution: translate firewall mark rules to linux traffic control `tc` commands


```
iptables -A PREROUTING -p udp -m udp \
 --sport 123 -j MARK --set-xmark 0x10/0x1f0
iptables -A PREROUTING -p udp -m udp \
 --dport 123 -j MARK --set-xmark 0x10/0x1f0
```


Linux and Inbound Shaping

```
iptables -A PREROUTING -p udp -m udp \
 --sport 123 -j MARK --set-xmark 0x10/0x1f0
```

This is translated to

```
tc filter add dev eth0 protocol ip \
 parent ffff: prio 35 basic match  \
 'u32 (u8 0x11 0xff at 0x9) and    \
  (u32(u16 0x7b 0xffff at 0x14))'  \
 action ipt -j MARK --set-xmark 0x10/0x1f0 \
 action mirred egress redirect dev ifb0
```

Ugly? Yes. But it works.

Linux and Inbound Shaping: Re-Classification


```
iptables -A PREROUTING -m mark --mark \
 0x20/0x1f0 -m length --length 1000:65535 \
 -j MARK --set-xmark 0x80/0x1f0
```

This is translated to

```
tc filter add dev ifb0 protocol ip \
 parent 1: prio 7 basic match \
 'meta(fwmark mask 0x1f0 eq 0x20) \
 and meta(pkt_len gt 999)' flowid 1:6
```

Linux and Inbound Shaping: TC-Translator

- Normal mark rules are translated to `ipt -j MARK` action + `mirred egress redirect` to `ifb`-device
- Re-Classification rules are translated to flowid statement *in the `ifb`-device* to put the packet into the correct queue
- Rules from `iptables` are processed *in reverse order*
- remember: Rules in `tc` are terminating, rules in `iptables` are not

Linux and Inbound Shaping

To sum up:

- We do traffic shaping just by specifying appropriate rules in the fwbuilder GUI
- The dirty work is done behind the scenes:
  - classification of traffic according to firewall marks for outbound traffic
  - translation of firewall marks to appropriate `tc` commands and redirection to `ifb` device for inbound shaping

Help

Setting the “nexthdr” pointer for IPv4 in `tc`

- I’ve never figured out how to correctly determine the offset of the “next protocol” (i.e. UDP or TCP) in the presence of IP options with the `tc` basic match
- I’m aware how to do this using hash tables and `u32`
- but hash tables are unusable with basic match
- others have already called this extremely non-obvious

Availability

Get scripts from rsclib.sourceforge.net

```
% python
from rsclib import trafficshape
help (trafficshape)
```

or see →16

Bibliography

[Bro06] Martin A. Brown. Traffic control howto. Howto, Linux IP, October 2006.

[CJOS01] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith. Tuning RED for web traffic. *IEEE/ACM Transactions on Networking*, 9(3):249–264, June 2001.

[FJ93] Sally Floyd and Van Jacobson. Random early detection gateways for congestion

Bibliography

avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[Flo97] Sally Floyd. Discussions of setting parameters. Email, International Computer Science Institute (ICSI) Networking Group (ICIR), 1997.

[Nag85] John Nagle. On packet switches with infinite storage. RFC 970, Internet Engineering Task Force, December 1985.

Bibliography

[Nag87] John Nagle. On packet switches with infinite storage. *IEEE Transactions on Communications*, 35(4):435–438, April 1987.

[Sch09] Michael Schwartzkopff. Howto manage traffic shaping with fwbuilder. Howto, MultiNET Services GmbH, April 2009.

[SZN00] Ion Stoica, Hui Zhang, and T. S. Eugene Ng. A hierarchical fair service curve algorithm for link-sharing, real-time and priority

Bibliography

services. *IEEE/ACM Transactions on Networking*, 8(2):185–199, April 2000.