Replacing HTB with EDT and BPF

Stanislav Fomichev, Eric Dumazet, Willem de Bruijn, Vlad Dumitrescu, Bill Sommerfeld, Peter Oskolkov

Google
Mountain View, CA
sdf@google.com, edumazet@google.com, willemb@google.com, vladum@google.com, wsommerfeld@google.com, posk@google.com

Abstract

Traditionally, rate limiting on Linux has been done using HTB qdisc. Flow aggregates are classified into buckets and a token-bucket mechanism is used to ensure the proper distribution of bandwidth among multiple buckets.

Recently, the Linux TCP stack switched to the Early Departure Time model [1]. With this model, every packet flowing through the stack has a departure timestamp which can be adjusted. This makes it possible to implement rate limiters in BPF.

This paper shares the details on one possible BPF rate limiter implementation.

Keywords

Linux, TCP, EDT, EBPF, BPF, HTB, QDISC.

Introduction

Google servers classify, measure, remark (DSCP), and rate limit their outgoing traffic. Historically, all four functionalities were implemented using an HTB hierarchy with filters and actions [2]. Over time, classification, measurement and remarking were moved to the TC egress BPF hook, but rate limiting still used HTB.

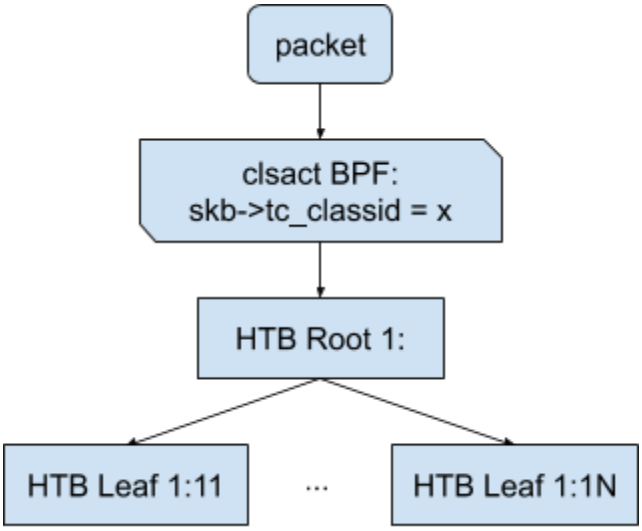
At netdev 0x12, Van Jacobson proposed to replace networking queues with a timing wheel and Earliest Departure Time (EDT) in every skb, and this has been eventually implemented in Linux [3]. Google's rate limiting logic uses dynamically adjusted rates that avoid congestion on the link. As a result, rate limiting can be implemented by adjusting the EDT timestamps as desired using BPF code, and relying on the kernel's FQ schedulers to release packets at the time specified by the EDT timestamp.

Motivation

Google uses a reactive control system called BwE to push flow aggregate rate limits on demand [4]. This means that most of the egress traffic at Google doesn't have a rate limit. Through some private patches, this traffic was bypassing the HTB. Even though, percentage wise, only a small amount of traffic was rate-limited, on the modern NICs it's still a considerable amount. These flows still had to traverse HTB, contending on the global root qdisc lock.

Design Overview

When we moved part of traffic management functions to BPF, the kernel did not support EDT skb timestamps. As a consequence, rate limiting still had to be implemented by HTB.



In detail, the program at the TC egress BPF hook did the following:

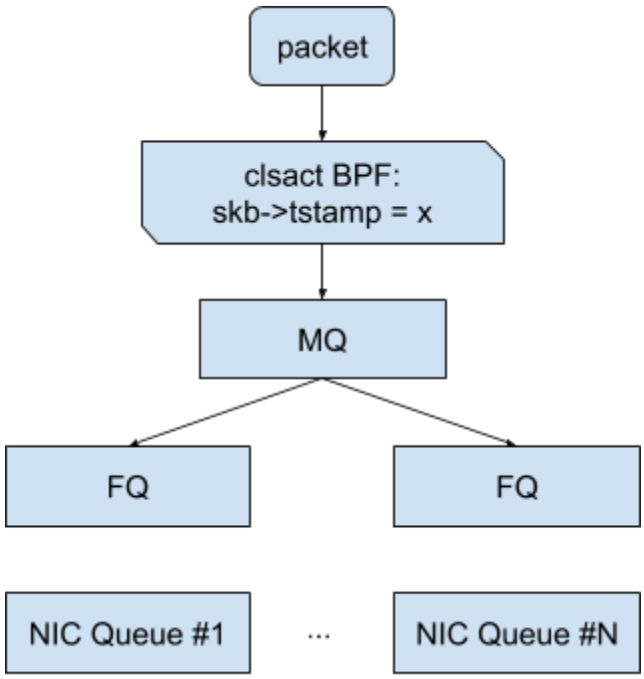
- Classify the packet into the flow aggregate (essentially, figure out which flow belongs to which container).
    - If classified flow aggregate doesn't have a rate limit, set a special skb field that indicates HTB bypass.
    - If classified flow aggregate has a rate limit, set skb's tc_classid to the appropriate value. HTB will do the rate enforcement.
- Measure throughput and number of packets.

- Sometimes rewrite DSCP bits to change traffic QoS.

Following that, each rate-limited flow aggregate was assigned to a specific HTB leaf node with appropriate rate limit, implemented using the following, flat hierarchy of HTB classes:

Replace HTB with BPF+FQ

Once TCP stack was converted to EDT model and appropriate skb fields were exported to BPF we've switched to the following model:



HTB qdisc was replaced with a set of FQ qdiscs (one per NIC queue) which provides per-flow fairness and enforcement of skb departure timestamp. This eliminated contention on the global HTB lock and the global qdisc lock now is split into per-flow-aggregate state. Our BPF program was extended with a per-flow-aggregate map and a small amount of BPF was written to keep track of the per-low-aggregate state.

Here is an example of simple rate-limiter in BPF:

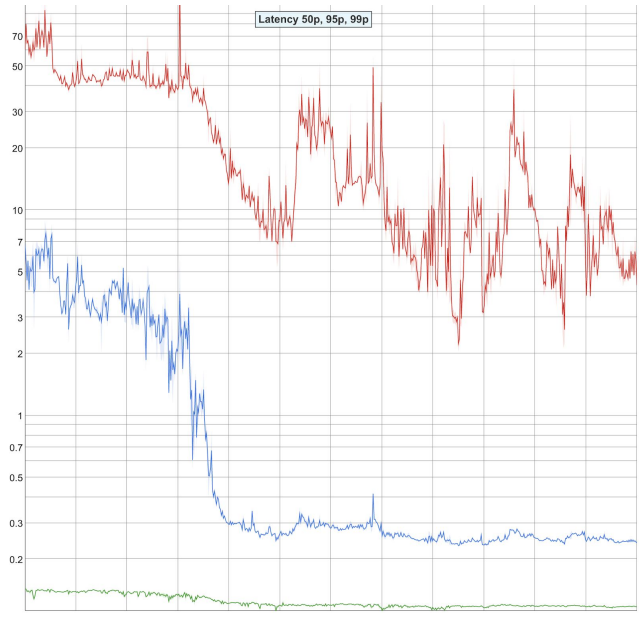
```
# classify packet into flow aggregate
aggregate_state = state[classify(skb)]

delay_ns = skb->len * NS_PER_SEC /
aggregate_state->rate_limit_bps
next_tstamp = &aggregate_state->next_tstamp

if *next_tstamp <= now:
   # racy, not an issue, same value expected
   *next_tstamp = now + delay_ns
   return TC_ACT_OK

if *next_tstamp - skb->tstamp >= DROP_HORIZON:
   # DROP_HORIZON is 2s
   return TC_ACT_SHOT

if *next_tstamp > skb->tstamp:
   skb->tstamp = *next_tstamp # rate-limit

__sync_fetch_and_add(next_tstamp, delay_ns)
return TC_ACT_OK
```

Evaluation

Apart from flexibility, the switch from HTB to BPF+FQ replaces the expensive single-lock contention with cheaper _sync_fetch_and_add() contention. This gave us significant performance benefits in both CPU utilization, and transmission latency.

The following graphs clearly show the improvement in the transmission latency for the rate-limited flow aggregates (Y axis is normalized to the range of 0 to 100).

50% (green), 95% (blue) & 99% (red) TX Latency:



There is about 20x improvement in 95% latency and 10x improvement in 99% latency.

Conclusion

Migrating from HTB to BPF+FQ for traffic shaping significantly improved tail latency and simplified our setup on the hosts. Having a rate limiter in BPF also allows us to iterate faster, roll out new features and fix bugs.

Acknowledgements

Special thanks to Eric Dumazet for coming up with the EDT approach. Thanks to Peter Oskolkov for the initial implementation. Thanks to Vlad Dumitrescu and Bill Sommerfeld for working on productionization, rollout and impact summary. Thanks to everyone else who provided the comments and suggestions.

References

1. Eric Dumazet, *[PATCH net-next 0/9] tcp: switch to Early Departure Time model*
2. Vlad Dumitrescu, *Scaling Linux Traffic Shaping with BPF*
3. Van Jacobson, *Evolving from AFAP: Teaching NICs about time*
4. *BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing*