Scaling Linux Traffic Shaping with BPF

Linux Plumbers Conference 2018
BPF Microconference

November 15, 2018
Vancouver, BC, Canada

work by (alphabetically):

Chonggang Li, Craig Gallek, Eddie Hao,
Kevin Athey, Maciej Żenczykowski,
Vlad Dumitrescu, Willem de Bruijn,
Xiaotian Pei, and many others at Google

presented by: *Vlad Dumitrescu*

Outline

1. **Context**; HTB solution; problems with HTB.
2. **BPF** to the rescue! And other advantages.
3. Guided, but open **discussion** on BPF related issues.

Context

Servers **classify**, **measure**, **rate limit** and **remark** (QoS) outgoing traffic.

Traffic Control (TC) hierarchy w/ HTB & dsmark qdiscs; u32 and custom filters.

Userland daemon

maintains TC hierarchy, collects statistics, communicates with control system (BwE[1])

Classification parameters

dst cluster, src container, QoS, delegated user[1] (a custom socket option and skb field)

[1] *Alok K., et al. BwE: Flexible, Hierarchical Bandwidth Allocation for WAN Distributed Computing (doi.org/10.1145/2785956.2787478)*

Problems[2]

- very large TC HTB tree
  - sometimes 25k nodes
  - slow stats collection
  - per packet costs

- kernel changes/rollout for new needs
  - custom filters, optimizations

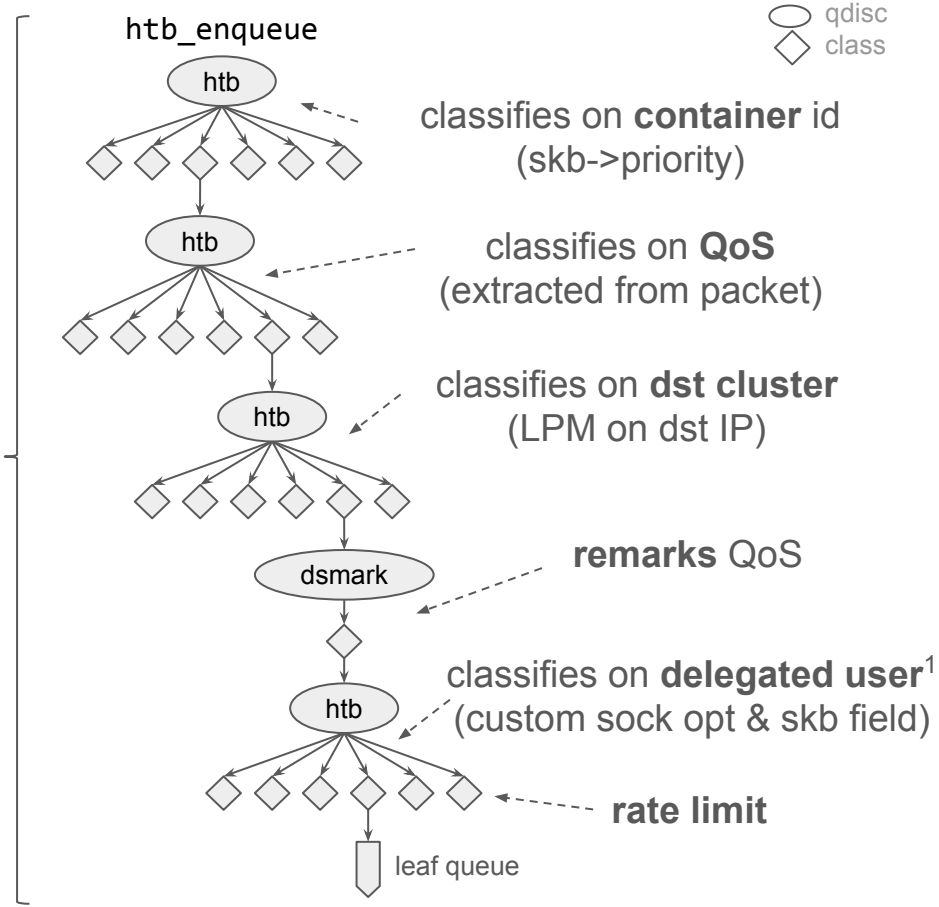
Over time, a lot of traffic (e.g., intra-cluster) ended up **bypassing** this (custom sock opt & skb flag).

[2] also tackled, for some environments, in: A. Saeed, et. al. Carousel: Scalable Traffic Shaping at End Hosts (doi.org/10.1145/3098822.3098852)

Problems[2]

- very large TC HTB tree
  - sometimes 25k nodes
  - slow stats collection
  - per packet costs
- **the dreaded root qdisc lock**
- kernel changes/rollout for new needs
  - custom filters, optimizations

Over time, a lot of traffic (e.g., intra-cluster) ended up **bypassing** this (custom sock opt & skb flag).



[2] also tackled, for some environments, in: A. Saeed, et. al. Carousel: Scalable Traffic Shaping at End Hosts (doi.org/10.1145/3098822.3098852)

BPF to the Rescue

Classify

BPF on TC clsact egress. Builds *flow key*.

cluster (LPM on dst IP), QoS, container (skb field), delegated user (skb field)

Measure

Increment `{bytes, packets}` for *flow key* (per-CPU map).

Remark (QoS)

Rules for *flow key* in BPF hash map. Change `skb->tos`.

Rate Limit

Set *classid* or *bypass* based on rules. Still HTB, but flat.

At least **95% of traffic** is **not rate limited**

=> gets accounted, but **bypasses** HTB

=> **qdisc root lock no longer matters**.

BPF to the Rescue

Classify

BPF on TC clsact egress. Builds *flow key*.

cluster (LPM on dst IP), QoS, container (skb field), delegated user (skb field)

Measure

Increment `{bytes, packets}` for *flow key* (per-CPU map).

Remark (QoS)

Rules for *flow key* in BPF hash map. Change `skb->tos`.

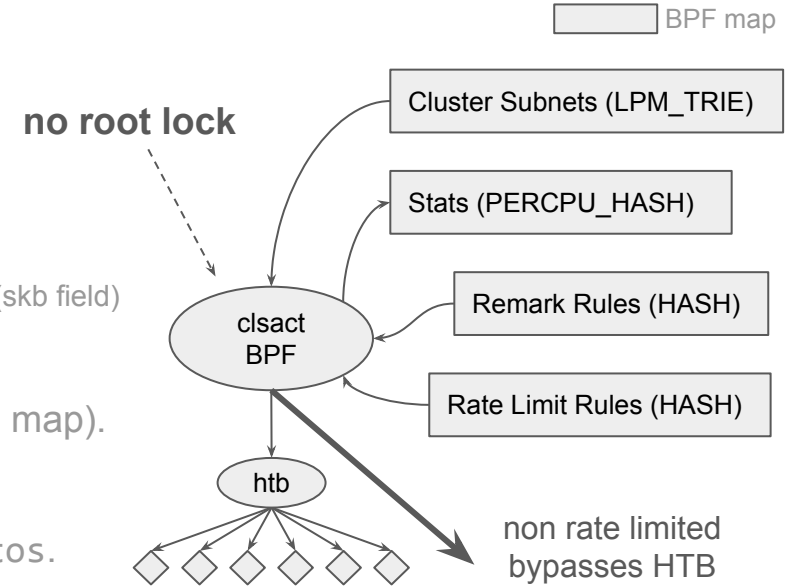
Rate Limit

Set *classid* or *bypass* based on rules. Still HTB, but flat.

At least **95% of traffic** is **not rate limited**

=> gets accounted, but **bypasses** HTB

=> **qdisc root lock no longer matters.**



Could be replaced in the future.
We don't need token borrowing.
Only **flat space of queues,** which could be lock-free or fine-grained.

Custom patch to allow bypass of root qdisc after clsact.

Other Advantages of BPF

- dynamic socket-level policies
  - congestion control, but also other socket options
  - using TCP-BPF, which runs on TCP socket state transitions (i.e., passive/active established)
- *first-packet* classification
  - delay in building HTB hierarchy (add/remove/change nodes) as flows appear & rules change
  - with BPF, some rules are always configured, and will apply to the *first packet*
- faster deployment of business logic changes, bug fixes
- optimization opportunities
  - replace hot paths (e.g., map/trie lookups) with generated BPF instructions

Open Discussion (1/2)

1. complete map dumps from userland
   - 2 syscalls (`BPF_MAP_LOOKUP_ELEM`, `BPF_MAP_GET_NEXT_KEY`) per entry
   - for *Stats* (`PERCPU_HASH`), we always read all items, every N seconds
2. better Longest Prefix Match trie implementation
   - `trie_lookup_elem` in top 5 kernel CPU users in our continuous profiling[3]
3. runtime map resizing controlled by userland
   - *Stats* `PERCPU_HASH` map provisioned for worst case
4. limited unit testing capabilities
   - `bpf_prog_test_run` only allows fake data, not fake `skb`/`__sk_buff`
5. bypass root qdisc after clsact egress (via TC_ACT_* return code?)

[3] *Gang R., et al. Google-Wide Profiling: A Continuous Profiling Infrastructure for Data Centers (doi.org/10.1109/MM.2010.68)*

Open Discussion (2/2)

1. memory management for (per-CPU) maps
   - allocation pattern makes the per cpu allocator reach a highly fragmented state
   - sometimes takes a long time (up to 12s) to create the `PERCPU_HASH` maps at startup
2. performance and profiling
   - always-on CPU usage for each program instance
   - kprobes inside BPF programs
3. hidden Direct Packet Access write overheads
   - verifier decides to always `bpf_skb_pull_data` if program has DPA writes
   - workarounds: call DPA writing program only when needed; use `bpf_skb_store_bytes`
4. plumbing a new field to `__sk_buff` requires kernel changes
   - verifier as a module?
   - *or maybe this is why a "Plumbers" conference exists :)*

Thank You!

Eddie Hao <eddieh@google.com>
Vlad Dumitrescu <vladum@google.com>
Willem de Bruijn <willemb@google.com>