Google

Replacing HTB* with EDT and BPF

Netdev 0x14, March 2020

* HTB rate limiting

Context

- Our servers **classify**, **measure**, **rate limit** and **remark** (QoS) outgoing traffic.
- At LPC2018 we've talked about how we had moved **classify**, **measure** and **remark** parts into BPF programs.
- This is a follow up work on how we moved **rate liming** into BPF as well and removed HTB.

Define EDT and FQ

EDT - Earliest Departure Time - proposed by Van Jacobson at netdev 0x12.

- Repace queues with timer wheel
- Each skb has minimum departure time (tstamp)

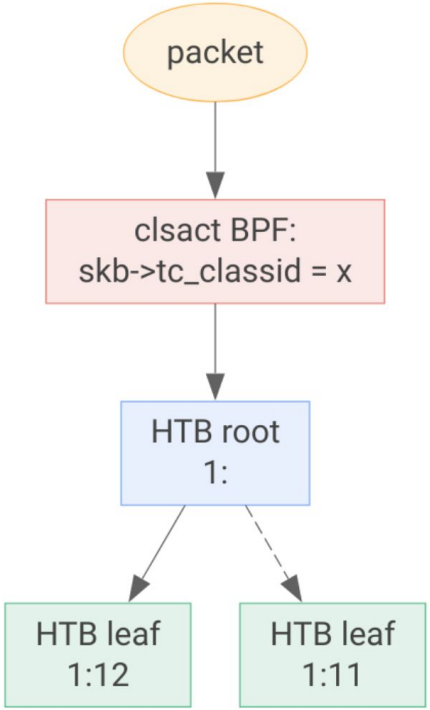
FQ - Fair Queue Packet Scheduler - maintains per-flow fairness.

- Per NIC-queue, no global state/lock
- Per flow RB-tree, RR between flows

Linux TCP stack and FQ qdisc have been switched to EDT in Sep 2018.

Pre-EDT architecture

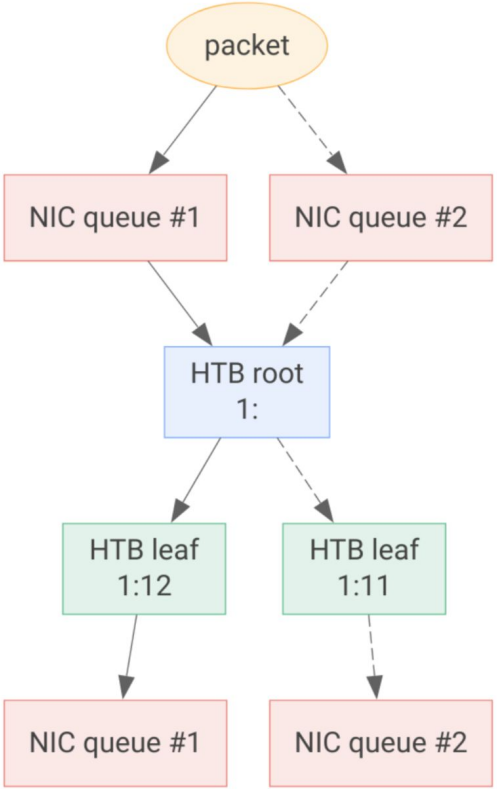
- Rate-limiting rules are dynamically pushed onto machine by a control system called BwE
- Daemon on the machine installs leaf nodes in HTB *flat* hierarchy and adjusts BPF maps
- BPF program classifies traffic by setting `skb->tc_classid`
- Most traffic, most of the time, is *not* rate-limited (bypasses HTB via private patches)
- When many flows need to be rate limited on a host, we still get occasional contention in HTB



How HTB works with multi-queue NICs

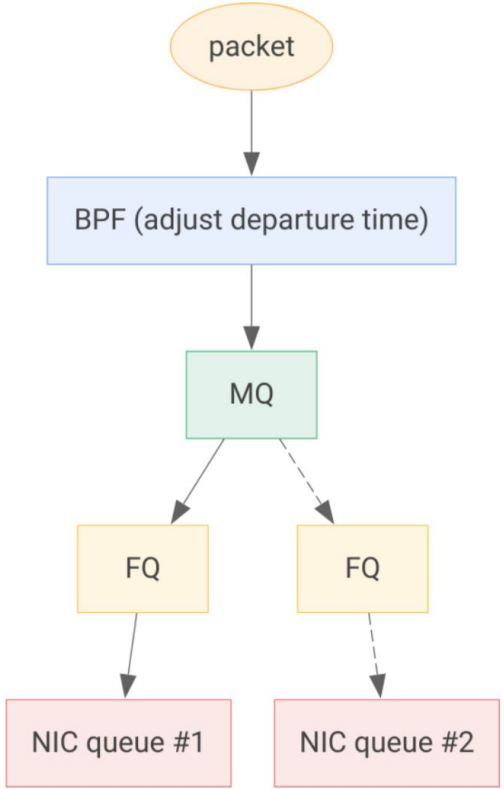
HTB sits in front of *all* NIC queues and each packet enqueue takes root lock:

```
dev_queue_xmit():
 txq = netdev_core_pick_tx(dev, skb,...)
 # qdisc points to the same HTB for all queues
 qdisc = txq->qdisc
 __dev_xmit_skb(skb, qdisc, dev, txq):
   spin_lock(qdisc.lock) # qdisc root lock
   qdisc->enqueue(skb, ...) # classify and queue
   spin_unlock(qdisc.lock)
```



How can we fix it?

- Don't use HTB, use FQ instead
- Adjust `skb->tstamp` from BPF when rate limiting
- FQ will ensure that packet is not sent out until `now >= tstamp`
- Use MQ to create a per-queue FQ



BPF reference implementation

```
aggregate_state = state[classify(skb)] # classify packet into flow aggregate
delay_ns = skb->len * NS_PER_SEC / aggregate_state->rate_limit_bps
next_tstamp = &aggregate_state->next_tstamp

if *next_tstamp <= now:
   *next_tstamp = now + delay_ns # racy, not an issue, same ballpark value expected
   return TC_ACT_OK
if *next_tstamp - skb->tstamp >= (int64_t)DROP_HORIZON: # 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
```

Limitations

Not a complete HTB replacement, we target per-queue rate limiting:

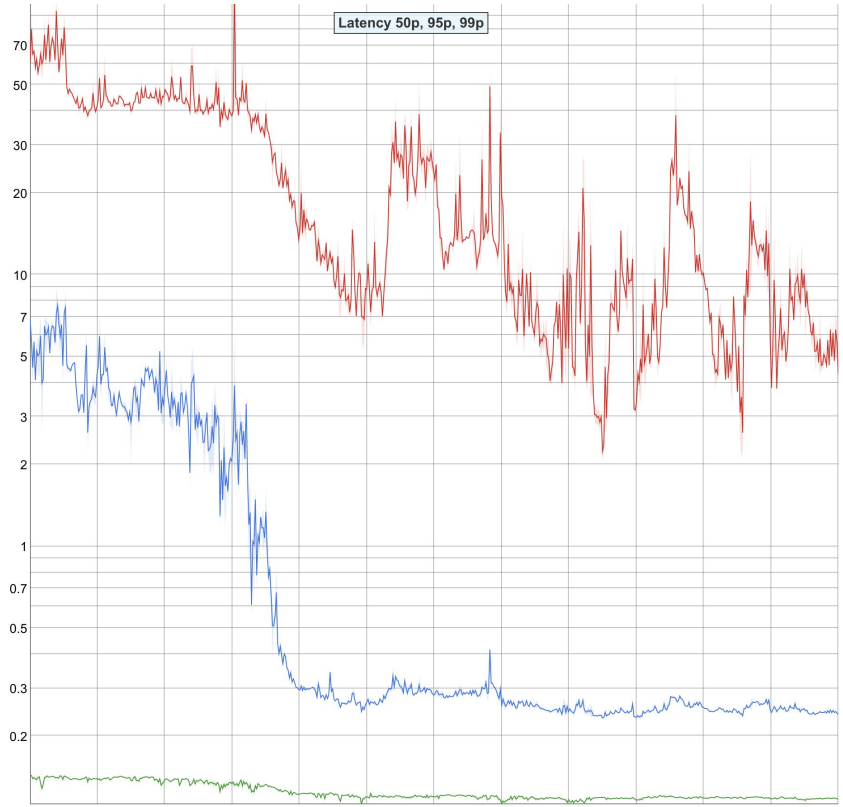
- No global locking required to maintain shared state
- Scheduling is deferred to FQ

Out of scope:

- HTB is work conserving (classes can borrow from one another)
- HTB classes can be assembled into hierarchy

Can this all be done with BPF? Probably yes.

Evaluation



More examples

- tools/testing/selftests/bpf/progs/test_tc_edt.c
- samples/bpf/hbm_edt_kern.c

Questions?