Hierarchical QoS Hardware Offload (HTB)

Yossi Kuperman, Maxim Mikityanskiy, Rony Efraim

Mellanox Technologies Ltd.
Yokneam, Israel
{yossiku,maximmi,ronye}@mellanox.com

Abstract

Traffic shaping is essential to a correct and efficient operation of datacenters. To name a few examples, policy-based bandwidth allocation to a group of flows, and packet pacing to reduce traffic burstiness that can overwhelm router buffers. As network bandwidth demand increases so does the number of classified flows, exposing scalability limits of the HTB implementation.

HTB is quite flexible and versatile, but it comes with a cost. HTB does not scale (due to a single lock) and consumes considerable CPU and memory. In our proposed solution, classification takes place in software, whereas rate-limiting is done by the hardware. By moving the classification to clsact egress hook, which is thread-safe and does not require locking, we avoid the contention induced by the single qdisc lock. Furthermore, offloading the shaping logic and the maintenance of the intra-class bookkeeping into the hardware, completely removes the need for data-path synchronization. Our aim is to offload HTB functionally to hardware, providing the user with the flexibility and the conventional tools offered by TC subsystem while scaling to thousands of traffic classes and maintaining wire-speed performance.

Keywords

Linux, queuing disciplines (qdisc), quality of service (QoS), hierarchical token bucket (HTB), network offload

Introduction

Traffic shaping is required for efficient operation of production networks. However, it uses CPU cycles on data-center servers that can otherwise be used for running user applications.

HTB queueing discipline (qdisc) allows the use of a physical link to simulate several slower links---represented by classes. This is done by configuring a hierarchical QoS tree. Each tree node corresponds to a class; there are inner classes and leaf classes. Inner classes represent the hierarchal structure and the borrowing relationships; traffic shaping takes place at the leaf classes. Filters are used to classify flows to the different classes.

HTB is quite flexible and versatile, but it comes with a cost. HTB does not scale properly and consumes valuable CPU cycles to implement traffic shaping. We aim to offload HTB functionally to hardware and provide the user with the flexibility offered by the HTB qdisc, while maintaining wire-speed performance.

Hardware Model

Mellanox hardware, more specifically, ConnectX-5 and upwards, provides a hierarchical quality of service offload capability. The firmware provides a mechanism to configure a QoS tree, where the leaf nodes are exposed to the driver through send queues (SQ). This SQ is a hardware queue with FIFO semantics. The hardware schedules the SQs according to their configured rate-limit. Rate-limiting is enforced by a token bucket shaper. The hierarchy is for borrowing between leaf nodes with a common parent.

The aforementioned description bears a resemblance to Linux's HTB queueing discipline. Note that flow classification still takes place in software. TC egress hook (clsact) is used to classify outgoing packets into one of several traffic classes, each is associated with a SQ, as depicted in Figure 1.

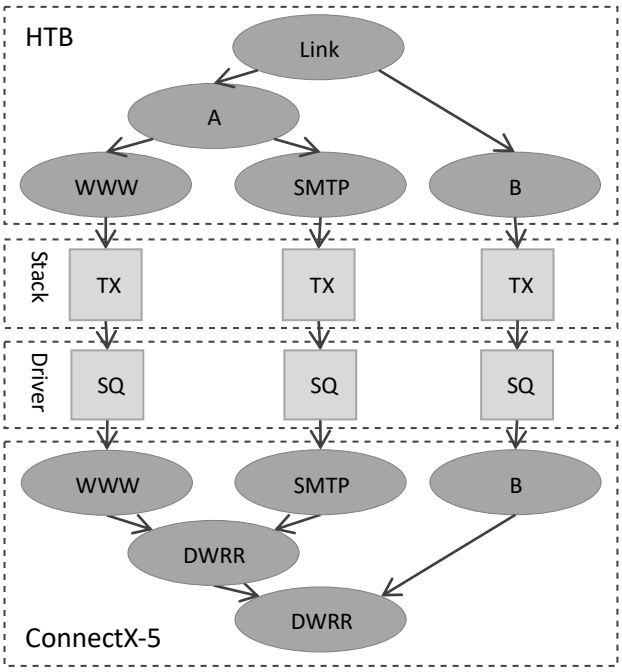

Figure 1 Hierarchical QoS Tree Example

Our Proposed Design

We would like to enhance the HTB qdisc to support offload. HTB provides us with the common tools to configure the desired QoS tree.

The most concerning aspect of the current HTB implementation is its lack of support for multi-queue. All TX queues of a netdevice point to the same HTB instance, resulting in high spin-lock contention. This contention (might) negates the overall performance gains expected by introducing the offload in the first place.

For each HTB leaf-class the driver will allocate a special send queue (SQ) and match it with a corresponding net-device TX queue. A unique FIFO qdisc will be attached to any such TX queue. Classification will still take place in software, but rather at the clsact egress hook. Once a packet is classified it will be directed to the corresponding TX/SQ that will do traffic shaping in hardware.

The rest of this section covers the relevant components we need to tackle in order to support HTB hardware offload.

Making HTB a Multi-Queue Qdisc

We should modify HTB to present itself as mq qdisc does. By default, mq qdisc allocates a pfifo qdisc (packet limited first in, first out queue) per TX queue exposed by the lower layer device. This only occurs when hardware offload is configured, otherwise, HTB behaves as usual. There is no HTB code along the data-path; the only overhead compared to regular traffic is the classification taking place at clsact.

HTB will notify the driver of its existence by means of .ndo_setup_tc().

This design induces full offload---no fallback to software. It is not trivial to partially offload the hierarchical tree considering borrowing between siblings anyway.

Create/Destroy New Traffic Class

Upon a new class creation/destruction, HTB will notify the driver using .ndo_setup_tc(). This way the user builds the QoS hierarchal tree; the driver should configure the hardware to mirror this tree.

Each leaf class is represented by a netdev TX queue, and each TX queue is backed by a special hardware send-queue (SQ). However, it might be challenging to create a TX queue on-the-fly by the driver. One possible way to circumvent this is to pre-allocate TX queues at the driver initialization. Upon a new class creation, the driver will allocate an SQ with the appropriate configuration and assign it to an empty TX queue. New TX queues are allocated by increasing the number of ‘real’ TX queues (real_num_tx_queues). While HTB is configured and offloaded, we should prohibit the user from manually changing the number of channels (e.g., ethtool -L).

Note: we do not know in advance which class is inner and which is a leaf; it is up to the driver implementation to handle it properly.

Packet Classification

HTB uses filters to classify packets to classes. Usually, filters are applied to the HTB qdisc. However, the selection of the TX queue occurs prior to the HTB classification. In order to derive the TX queue from the class we ought to perform the classification before TX queue is determined.

We can use clsact anchor point (fake qdisc), which happens before the TX queue selection, to perform the classification. HTB skips the classification if the “priority” field of the sk_buff structure, which is treated as classid, resolves to an existing class. We can slightly change the usual way filters are configured to HTB with equivalent filters using skbedit action as follows:

```
$ tc qdisc add dev eth0 clsact
$ tc filter add dev eth0 egress protocol ip flower dst_port
80 action skbedit priority 1:10
```

The classification done at the clsact egress point is lock-free, and thus being performed concurrently, whereas, HTB performs all the classifications under the same qdisc lock.

HTB supports hierarchical classes and each inner class can have filters attached. This provides the user the flexibility of configuring different layers of filters, for example, at the top class the user can classify by container-id and more specific classification will take place at inner classes. We lose this ability when using clsact. However, we can simulate multiple layers by different TC chains.

TX Queue Selection

Classification outcome is stored in sk_buff’s “priority” field. The network stack gives the driver a chance to select the TX queue by means of .ndo_select_queue(). The driver should know that HTB is configured and must use the “priority” as a hint. It is the driver responsibility to maintain a mapping between classes and TX queues. The driver translates class to TX queue and returns it to the network stack.

HTB uses an internal queue denoted by htb-direct. Packets are first drained from this queue; no QoS is applied to this queue. HTB enqueue packets to this internal queue when classification fails, or the chosen class is the handle of the root qdisc. We can provide similar behavior by selecting the classic TX queues that do not perform any QoS.

HTB Statistics

We should properly handle statistics request from the user. Upon such request, HTB qdisc must delegate the request to the driver. The driver will then inquire the firmware for the statistics per class.

References

[1] Daniel Borkmann. net, sched: add clsact qdisc
https://lwn.net/Articles/671458/

[2] HTB - Hierarchy Token Bucket
https://linux.die.net/man/8/tc-htb

[3] Jamal Hadi Salim, Linux Traffic Control ClassifierAction Subsystem Architecture, Proceedings of Netdev
0.1, Feb 2015

[4] Ahmed Saeed, Nandita Dukkipati, Vytautas Valancius, Vinh The Lam, Carlo Contavalli, and Amin Vahdat. Carousel: Scalable traffic shaping at end hosts. In Proceedings of the Conference of the ACM Special Interest Group on Data Communication, SIGCOMM '17. ACM, 2017