Google

From Queues to Earliest Departure Time

Nandita Dukkipati, Eric Dumazet, Van Jacobson, Amin Vahdat & ...

David Wetherall (presenter)

Oct 2018
Google Platforms and Infrastructure designs...

Datacenters
Computers
Networks
Software

... to turn a network of computers into a single global computer.
Scaling: Data Center Network Bandwidth Growth

Traffic generated by servers in our datacenters

Aggregate traffic

Time


50x

1x
Google Datacenter Network Innovation
And hardware scale that we could not buy
Google’s data center in Douglas County, GA
https://www.google.com/about/datacenters/
Google Network
More than a collection of data centers
Our distributed computing infrastructure required networks that did not exist.
Subset of Google networking publications

See [http://g.co/research/networkinfra](http://g.co/research/networkinfra)

<table>
<thead>
<tr>
<th>Conference</th>
<th>Title</th>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIGCOMM 2018</td>
<td>B4 and After: Managing Hierarchy, Partitioning, and Asymmetry for</td>
<td></td>
</tr>
<tr>
<td>NSDI 2018</td>
<td>Andromeda: Performance, Isolation, and Velocity at Scale in Cloud</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2017</td>
<td>Carousel: Scalable Traffic Shaping at End-Hosts</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2017</td>
<td>Taking the Edge off with Espresso: Scale, Reliability and Programmability</td>
<td></td>
</tr>
<tr>
<td>CACM 2017</td>
<td>BBR: Congestion-Based Congestion Control</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2016</td>
<td>An Internet-Wide Analysis of Traffic Policing</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2016</td>
<td>Evolve or Die: High-Availability Design Principles Drawn from Failures in a</td>
<td></td>
</tr>
<tr>
<td>NSDI 2016</td>
<td>Maglev: A Fast and Reliable Software Network Load Balancer</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2015</td>
<td>TIMELY: RTT-based Congestion Control for the Datacenter</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2015</td>
<td>Condor: Better Topologies through Declarative Design</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2015</td>
<td>Bandwidth Enforcer: Flexible, Hierarchical Bandwidth Allocation for WAN</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2015</td>
<td>Jupiter Rising: A Decade of Clos Topologies and Centralized Control in</td>
<td></td>
</tr>
<tr>
<td>NSDI 2014</td>
<td>Libra: Divide and Conquer to Verify Forwarding Tables in Huge Networks</td>
<td></td>
</tr>
<tr>
<td>SIGCOMM 2013</td>
<td>B4: Experience With a Globally-Deployed Software Defined WAN</td>
<td></td>
</tr>
</tbody>
</table>
From Queues to Earliest Departure Time

Nandita Dukkipati, Eric Dumazet, Van Jacobson, Amin Vahdat & ... 
David Wetherall (presenter)

Oct 2018
TCP sends AFAP (as fast as possible)

TCP’s reliable delivery constrains how much is sent but not how fast.

- lead to an AFAP (as fast as possible)* output contract, with shaping usually implemented by device output queue(s).
- queue length limit or receive window determines inflight between protocol & device.
- ‘how fast’ is implicit in the queue drain rate, a constraint that is local & upstream of the wire

* a.k.a. work conserving
Traffic Shaping with Queues

Packet Sources: Socket Buffers in Host OS or Guest VM

Shaper (AFAP)

Multiple Token Bucket queues in shaper for bandwidth policies
AFAP helped TCP/IP speed up 10,000x

Moore's law (18 month doubling time)

(25 years of Ethernet evolution)
... but AFAP makes bottleneck run at 100%

Queuing theory says this is fragile. E.g., for M/D/1:
Mitigating Factors

Pain in the form of delay/loss explosion comes from running above the line rate at the bottleneck for too long. This is less of an issue if:

- bandwidth-delay products are small and/or
- there’s a fat-buffered router in front of every bottleneck and/or
- links from hosts to ToRs run slower than fabric

The first of these saved us until ~1995 then the second & third until ~2012.

Since then pain has been increasing.
After 2000, going faster got hard

Difficult to keep fabric switches 10x faster than server NICs
Pain Relief

If we have less space (queues) to work with then we need to rely more on time

● Determine what’s AFAP at the bottleneck and run at that rate ...

Examples:

● HULL (NSDI’12) – ‘Less is more: trading a little bandwidth for ultra-low latency’
● BwE (Sigcomm’15) – ‘Flexible, Hierarchical Bandwidth Allocation for WAN’
● FQ/pacing (IETF88’13) – ‘TSO, fair queuing, pacing: three’s a charm’
● Timely (Sigcomm’15) – ‘RTT-based congestion control for the datacenter’
● BBR (CACM v60’17) – ‘Congestion-based congestion control’
● Carousel (Sigcomm’17) – ‘Scalable traffic shaping at end hosts’
Moving from AFAP to EDT (earliest departure time)

AFAP isn’t working for us now because it’s local and our problems aren’t

We need a model that allows more nuanced control of packet spacing on the wire

- The EDT model is a great candidate for a replacement
- Specify the earliest departure time of packets to control release

The enforcement *mechanism* needs to be just in front of (or in) the NIC to enforce relationships between all outgoing packets.

- *Carousel* (details soon) is a great example of such a mechanism.
Traffic Shaping with Queues

Packet Sources: Socket Buffers in Host OS or Guest VM

Shaper (AFAP)

Multiple Token Bucket queues in shaper for bandwidth policies
Meet the needs of network policies & congestion control, e.g. TCP

- Pace packets
- Provide backpressure
- Avoid HOL blocking

Use CPU & memory efficiently
Challenges:

CPU cost of maintaining queues grows super-linearly with #queues.

Synchronization cost on multi-CPU systems is dominated by locking and contention overhead when sharing queues amongst CPUs.

Example 1: Hierarchical Token Bucket (HTB) Linux Queuing Discipline.
Challenges:

CPU cost of maintaining queues grows super-linearly with #queues.

Synchronization cost on multi-CPU systems is dominated by locking and contention overhead when sharing queues amongst CPUs.
We do **don't** need queues and their associated cost.

Using **time** as a basic construct gives us all the control we need, and at very low cost.
Carousel’s core idea is to replace a complex of slow, brittle, concatenated queues with two simple pieces:

1. An *Earliest Departure Time (EDT)* timestamp in every socket buffer (skb)
2. A timing-wheel* scheduler replacing the queue in front of (or in) the NIC.

*= see, for example, [Hashed and Hierarchical Timing Wheels](https://www.sosp.org/sosp90/papers/paper54.pdf), Varghese & Lauck, SOSP 87.
Design of Carousel

1) Single, O(1), time-indexed queue, ordered by packet departure timestamps
1) Single, $O(1)$, time-indexed queue, ordered by packet departure timestamps

2) Apply Backpressure
1) Single, O(1), time-indexed queue, ordered by packet departure timestamps

2) Apply Backpressure

3) One Shaper per Core
Life of a Packet in Carousel

Compute Earliest Departure Time based on shaping rate

Enqueue Packet in Timing Wheel

Dequeue packet and deliver completion

To NIC
A timing wheel does what a queue does (and more) but is faster

Timing wheel insert & delete is $O(1)$ like a queue but with a smaller multiplier:
- cache friendly (no pointer chains)
- RCU friendly (single slot to update)

Driver (or NIC) gets to choose ‘event horizon’ (wheel length) so can do BQL-like tuning for long enough to fill wire but short enough to not blow away caches.
A timing wheel does what a queue does (and more) but is faster

Packets that would be sent after event horizon can get TSQ-like callback when they can be sent or get an ETooFar.

This replaces TSQ and fixes problem of many simultaneous writers generating huge queues.

It also puts hard bounds # of active output bytes, increasing probability of L3 cache hits for systems that can DMA from L3.
Qdiscs become purely computational – no more intermediate queues.

→ driver gets to see all packets in its event horizon so can easily do informed interrupt mitigation, lazy reclaim, (wifi) endpoint aggregation...

→ sender learns packet send time on `send()` and can handle deadlines, seek alternatives, do phase correction...
In essence, timing wheel is an in-memory representation of how packets will appear on wire. It can represent almost any causal scheduling policy.

(Policies like ‘Maximize Completion Rate’ are impossible to express with rates but easy with timestamps so we can finally make transactions ‘fair’ without stupidly slowing everything down.)

Qdiscs accomplish more with less
Summary

It’s time to change the host network model for sending from AFAP to EDT
It’s a match for our modern need to control packet spacing on the wire
It’s more efficient and effective than complex arrangements of queues
It admits rich scheduling policies
We hope it will unleash a wave of innovation
We’re converts, and we hope that you are too!
Thank You. Questions?