Congestion Control in the Real World

Lecture @Stanford, 10/12/20

Nandita Dukkipati
nanditagoogle.com
Why care about Congestion Control in Practice

Congestion Control delivers excellent end-to-end network performance and isolation through coupled host / NIC / switch capabilities for sharing network capacity.
Network Bandwidth Sharing at Google

Swift[1], TCP - GCN[2] and BBR[3]

Per-flow congestion control.

BwE [4], B4 TE [5]

Centralized Control of Flow Aggregates over WAN.

Static Limits

BW configuration based on CPU cores, storage etc.

[1] Swift: Delay is Simple and Effective for Congestion Control in the Datacenter, SIGCOMM 2020
[2] DCTCP: Data Center TCP (DCTCP), SIGCOMM 2010
Congestion Control: A Fundamental Network Building Block

- **Loss/RTT/ECN/Bandwidth Measurement Engine**
  - ACKs

- **CWND and Rate Computation Engine**
  - Increase / Decrease based on congestion detection signals

- **CWND and Rate Enforcement**

Data flows from left to right, with arrows indicating the direction of data transmission. The diagram illustrates the process of congestion control, with the CWND and rate computation engine adjusting the CWND and rate based on congestion detection signals.
**Congestion detection signals**

- **End-to-end**
- **Packet loss**
- **Round-trip time**
- **Bandwidth**

**Explicit Feedback from Network**
- Explicit Congestion Notification
- Queue lengths and differentials
- Sojourn time
- Available bandwidth
- Link utilization

**Loss**
- TCP New Reno, Cubic

**Delay**
- Vegas, Fast, BBR*, Swift

*DCTCP, XCP, RCP, DCQCN, HPCC*

* Also uses ECN, and max. throughput
Algorithms and Heuristics

Starting behavior
Slow Start Exponential growth

Steady State Behavior
Additive Increase and Multiplicative Decrease (AIMD)
Adaptive increase and decrease

Faster Convergence
Hyper-active Increment
Cubic increase
Explicit Congestion Notification (ECN)

- Switches set “Congestion Experienced” bit on packets if the queue grows too large as per the IETF ECN standard.
- Switches inform receiver, which in turn can inform sender of congestion marks.
Datacenter TCP (DCTCP)

- Datacenter DCTCP (SIGCOMM 2010) uses ECN marks.
- Switches mark CE bit in IP header if queue > 65KB.
- Receivers reflect marks to senders (via TCP flags).
- Sender slows down according to proportion of marked packets each RTT.

\[ a \leftarrow (1 - g) \times a + g \times F \]

\[ cwnd \leftarrow cwnd \times (1 - \frac{a}{2}). \]
Reactive and Proactive Schemes

**Reactive Schemes**
Act on feedback gathered from acknowledgements.

**Proactive Schemes**
Proactively schedules network transfers.

*Centralized schemes* arbitrate globally for network transfers.

*Switch based schemes* explicitly allocate resources.

*Receivers* explicitly schedule transfers.
Metrics in evaluating Congestion Control

Application / User centric
- Response time of application’s data unit (flow-completion time, RPC completion time)
- Quality of experience for Video traffic
- Round-trip delay
- End-to-end goodput

Network Centric
- Queue delay
- Link throughput/utilization
- Buffer overflows
- Stability
## Congestion Control Challenges in Datacenter

### Congestion control requirements
- Transfers must complete quickly, low tail latency.
- Deliver high bandwidth (>> Gbps) and low latency (<< ms).
- Efficient use of CPU.

### Challenges
- Bursty traffic because of applications and NIC offloading.
- Small buffers.
- Very small round-trip delays.
- Incast traffic patterns with many (>1K) flows sharing very short paths.
- Kernel-bypassed transports.

### Opportunities
- Hardware assistance.
- Less worries of interoperability with legacy.
- Explicit network feedback is easier to deploy.
- Centralized control is possible.
Congestion Control Challenges in Wide Area Networks

High signal variability.

Small buffers and large round-trip times.

Mismatch in transport design and underlying link layer channel, e.g., channel bandwidth is time-varying and unpredictable, deep per-user buffers, burst scheduling algorithms

Deployed congestion control algorithms are heterogeneous and unknown to senders.

Coexistence with legacy algorithms that are sensitive to packet loss.

Explicit feedback from network is rare, and difficult to deploy widely.
Swift: Delay is Simple and Effective for Congestion Control in the Datacenter
What is Swift?

Swift is a delay based congestion-control for Datacenters that achieves low-latency, high-utilization, near-zero loss implemented completely at endhosts supporting diverse workloads like large-scale incast across latency-sensitive, byte and IOPS-intensive applications working seamlessly in heterogeneous datacenters with minimal switch support.

Swift achieves ~50μs tail latency for short-flows while maintaining near 100% utilization even at 100Gbps line-rate.
Why we built a new Datacenter congestion-control at Google?

New applications w/ low-latency requirements

- 100μs access latency at 100k+ IOPS for Flash
- NVM needs 10μs latency at 1M+ IOPS
- Large-scale incast for partition-aggregate workloads
- IOPS intensive applications, e.g., BigQuery shuffle operation

New stacks and new sources of congestion

- New networking stacks such as PonyExpress[1] exhibit different congestion behavior which is no longer limited to the fabric
- E.g., endpoint congestion becomes key for a non-interrupt based stack like PonyExpress

Increasing line-rates and robustness to heterogeneity

- 100Gbps networking and beyond
- Fast reaction to congestion - queue build-up happens very quickly

Design

Key aspects of Swift’s design
Swift in the context of PonyExpress

- command & completion queues
- op streams
- op scheduler
- Application

- flow mapper
- flows
- RTT
- CWND
- Pacing component
- Swift

- NIC queues
- CWND computation
- Delay computation
- Packet-level transport layer
- NIC
Swift Design

End-to-end delay decomposition of a Packet and its ACK

Swift maintains **two congestion-windows** (in #packets) - one based on fabric-delay and one based on endpoint-delay with their respective cwnd

Effective cwnd is the **minimum** of the two
### Swift Design contd.

<table>
<thead>
<tr>
<th>Simple AIMD based on a target-delay</th>
<th>Use of HW and SW timestamps</th>
<th>Seamless transition b/w cwnd and rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>if delay &lt; Target</td>
<td>To provide accurate delay measurements and separate them into fabric and host components</td>
<td>Swift allows cwnd to fall below 1 to handle large-scale incast</td>
</tr>
<tr>
<td>increase cwnd (Additively)</td>
<td></td>
<td>cwnd &lt; 1 implemented via pacing using Timing Wheel, pacing off when cwnd &gt; 1</td>
</tr>
<tr>
<td>else</td>
<td></td>
<td></td>
</tr>
<tr>
<td>decrease cwnd (Multiplicatively)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Swift Design contd.

Scaling of target-delay | Loss recovery and ACKing policy | Coexistence via QoS

Topology-based scaling (TBS) for RTT-fairness | Minimal investment in loss-recovery - losses are rare: SACK and RTO. | Multiple CC in shared deployments, e.g., WAN traffic, Cloud traffic etc.

Flow-based scaling (FBS for fairness)
Average Queue Buildup with Randomized flow arrival and perfect rate control
Key Takeaways

From experiences with deployment at Google
Production Results - Loss and Latency

Loss rate vs. Port Utilization at Edge

Latency vs. Cluster Throughput

Takeaways

Swift keeps loss-rates very small even at the 99.9th-p and at near line-rate utilization.

Loss-rate improvement doesn’t come at the cost of throughput; Swift sustains same cluster throughput as GCN.

We find that Swift is able to maintain the average fabric round-trip around the configured target delay.

GCN is a DCTCP\cite{2}-style congestion-control deployed at Google and serves as the comparison point for the production results presented here.

\cite{2} Datacenter TCP (DCTCP), SIGCOMM 2010
Production Results - Isolation in shared deployments

Isolation via QoS

Takeaways

Use of QoS works extremely well in providing isolation from non-Swift traffic in shared infrastructure.

Swift loss rate on the lowest priority QoS is lower than GCN loss rate on strict priority QoS.
Production Results - Endpoint congestion

Separation of Fabric vs. Endpoint Congestion

Throughput-intensive cluster

IOPS-intensive cluster

Takeaways

Endpoint congestion (measured by endpoint delays such as in the NIC Queue) is also important to address.

NIC delays can account for a significant portion of RTT, especially for IOPS intensive applications.
Key conclusions from our experiences with Swift deployment

**Delay works really well**
- Use of delay as a multi-bit congestion signal has proven effective for excellent performance
- Use of absolute target delay is performant and robust
- And simplicity that has helped greatly with operational issues.

**Fabric and Host congestion are both important to respond to**
- Both forms matter across a range of workloads.
- Delay is decomposable to separate concerns
- Important for end-to-end performance for applications

**Wide range of workloads**
- Including large scale incast
- Pace packets when there are more flows than the bandwidth-delay product (BDP)
- Use a congestion window at higher flow rates for CPU efficiency

**Wide range of workloads**
Future Directions for Research
Some immediate avenues for future work on Swift

What is the best possible delay based congestion control algorithm?
Fast convergence under bursty congestion
Analytical fluid model and exploration of control loop dynamics
How can we tell if Congestion Control is work conserving at Scale?
What is the **optimal increase function** for e2e Congestion Control?

Decrease is easier as it’s performed based on an explicit signal such as RTT or ECN.
A **systematic** way to handling **bottlenecks and congestion** at **hosts**
Congestion Control that can run in **Hypervisors** w/o direct access to **Guest transports**
Achieving ultra low latencies (<10us) for short transfers that’s close to propagation delay in the presence of bandwidth intensive transfers
Is Congestion Control at the packet layer fundamentally better than one at higher level entities such as messages (RMAs, RPCs)?
A robust well-performing and simple congestion control for the **WAN** that’s tolerant of *noisy signals* and works for small or large buffers.
Questions and Discussion

nanditad@google.com