Paper-Reading Report

Paper:

DCTCP, DCQCN, TIMELY, Swift,

HPCC: High-precision Congestion Control

Cebinae: Scalable In-Network Fairness Augmentation

Reporter: Senj Lee

Date: 1 st / July / 2024

Summary of Different CC Protocols

DCTCP

Overview

DCTCP combines Explicit Congestion Notification (ECN) with a novel control scheme at the sources. It extracts multibit feedback on congestion in the network from the single bit stream of ECN marks. Sources estimate the fraction of marked packets, and use that estimate as a signal for the extent of congestion. This allows DCTCP to operate with very low buffer occupancies while still achieving high throughput.

It is important to note that the key contribution here is not the control law itself. It is the act of deriving multi-bit feedback from the information present in the single-bit sequence of marks.

DCTCP uses ECN feedback from congested switches for AQM approaches, and the same applies to DCQCN.

Innovation points:

- DCTCP uses a simple marking scheme at switches that sets the Congestion Experienced (CE) codepoint of packets as soon as the buffer occupancy exceeds a fixed small threshold - K.
- Standard TCP cuts its window size by a factor of 2 when it receives ECN notification. In effect, TCP reacts to presence of congestion, not to its extent 2. Dropping the window in half causes a large mismatch between the input rate to the link and the available capacity. Specifically, **controller at the sender**:
  - $\alpha \leftarrow (1-g) \times \alpha + g \times F$
  - $cwnd \leftarrow cwnd \times (1 - \frac{\alpha}{2})$
- DCTCP starts marking early and aggressively – based on instantaneous queue length.

Advantages:

DCTCP alleviates three impairments:

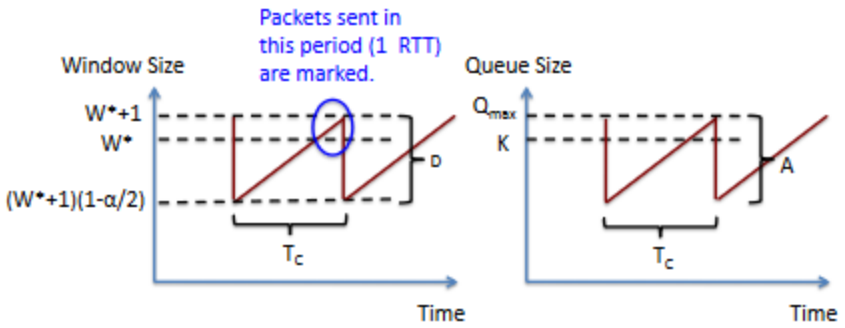
- **Queue buildup**: 发送方主动对拥塞程度的响应，可以保证交换机任意端口的队列长度不超过阈值 K ，从而减少了由于队列建立而产生的延迟，这对长流占用短流完成时间而产生的 HoL 阻塞很有帮助；
- **Buffer pressure**: 保留的缓冲区留空可以提高对微突发流的吸收能力，很大程度缓解包丢失问题；同样不会由于缓冲区压力过大而影响其他流的通过；

Disadvantages:

- **Incast**: If the number of small flows is so high that even 1 packet from each flow is sufficient to overwhelm the buffer on a synchronized burst, then there isn't much DCTCP—or any congestion control scheme that does not attempt to schedule traffic—can do to avoid packet drops. --> ***Swift*** !

Remaining problems -- why average queue buildup size is $\mathcal{O}(\sqrt{N})$? (I)

Queue Size at time t: $Q(t) = N \times W(t) - C \times RTT$, where,

- N: synchronized, long-lived flows with identical round-trip times RTT,
- $W(t)$: the window size of a single source at time t,
- C: shared single bottleneck link of capacity,
- 

Denote $S(W_1, W_2)$ is the number of packets sent by the sender, while its windows size increases from W_1 to W_2 . This takes $W_2 - W_1$ round-trip times, during which the average windows size is $\frac{W_1+W_2}{2}$,thus, $S(W_1, W_2) = \frac{W_2^2-W_1^2}{2}$.

Remaining problems -- why average queue buildup size is $\mathcal{O}(\sqrt{N})$? (II)

Critical window size at which the queue size reaches K : $W^* = \frac{C \times RTT + K}{N}$, and the switch starts marking packets with the CE codepoint. During the RTT it takes for the sender to react to these marks, its window size increases by one more packet, reaching $W^* + 1$.

Hence, the fraction of marked packets: $\alpha = \frac{S(W^*, W^*+1)}{S((W^*+1)(1-\frac{\alpha}{2}), W^*+1)}$, plugging $S(W_1, W_2) = \frac{W_2^2 - W_1^2}{2}$ into α, we get: $\alpha^2(1 - \frac{\alpha}{4}) = \frac{2W^*+1}{(W^*+1)^2} \approx \frac{2}{W^*}$. Instead, $\alpha \approx \sqrt{\frac{2}{W^*}}$.

As the previous figure depicted, the amplitude of oscillation in the windows size of a single flow, $D = (W^* + 1) - (W^* + 1)(1 - \frac{\alpha}{2})$. Since there are N flows in total,

- $A = ND = \frac{N(W^*+1)\alpha}{2} \approx \frac{N}{2}\sqrt{2W^*} = \frac{1}{2}\sqrt{2N(C \times RTT + K)}$,
- $T_C = D = \frac{1}{2}\sqrt{2\frac{C \times RTT + K}{N}}$,
- Finally, $Q_{max} = N(W^* + 1) - C \times RTT = K + N$.

DCQCN

Overview

Motivition: On IP-routed datacenter networks, RDMA is deployed using RoCEv2 protocol, which relies on PFC to enable a drop-free network.

> When the queue exceeds a certain threshold, a `PAUSE` message is sent by PFC to the upstream entity. The uplink entity then stops sending on that link till it gets an `RESUME` message.
>
> Hence, PFC can lead to poor application performance due to problems like head-of-line blocking, unfairness and victim flow.

The fundamental solution to PFC's limitations is a **flowlevel congestion control protocol**. In our environment, the protocol must meet the following requirements: (i) function over lossless, L3 routed, datacenter networks, (ii) incur low CPU overhead on end hosts, and (iii) provide hyper-fast start in the common case of no congestion. For example, QCN does not support L3 networks. Thus, DCQCN is proposed.

Innovation points:

- Extend QCN to IP-routed networks requires using the IP five-tuple as flow identifier, and adding IP and UDP headers to the congestion notification packet to enable it to reach the right destination;
- Do not demand any custom functionality from the switches, and the protocol is implemented in NIC.
- NP conveys this information back to the sender. The RoCEv 2 standard defines explicit Congestion Notification Packets (CNP) for this purpose. When an RP (i.e. the flow sender) gets a CNP, it reduces its current rate (RC) and updates the value of the rate reduction factor, α, like DCTCP, and remembers current rate as target rate (RT) for later recovery.
- There is no slow start phase. When a flow starts, it sends at full line rate, if there are no other active flows from the host. This design decision optimizes the common case where flows transfer a relatively small amount of data, and the network is not congested.

Advantages:

- Support Layer-3 networks.
- By providing per-flow congestion control, DCQCN **alleviates** PFC's limitations - unfairness, victim flow, poor efficiency, and etc. **But do not obviate the need for PFC**. With DCQCN, flows start at line rate. Without PFC, this can lead to packet loss and poor performance
- DCQCN is a rate-based congestion control scheme, because it is simple to implement than the window based approach, and allowed for finer-grained control.

Disadvantages:

- Making changes to the switches and NICs is especially problematic, as the QCN functionality is deeply integrated into the ASICs.
- DCQCN is not particularly sensitive to congestion on the reverse path, as the send rate does not depend on accurate RTT estimation

TIMELY

Overview

TIMELY can adjust transmission rates using RTT gradients to keep packet latency low while delivering high bandwidth.

> - **RTT directly reflects latency**.
> - **RTT can be measured accurately in practice**. Recent NICs provide hardware support for high-quality timestamping of packet events , plus hardware-generated ACKs that remove unpredictable host response delays.
> - **RTT is a rapid, multi-bit signal**.

TIMELY is the first delay-based congestion control protocol for use in the datacenter, and it achieves its results despite having an order of magnitude fewer RTT signals (due to NIC offload) than earlier delay-based schemes such as Vegas.

Innovation points:

Summary: TIMELY

- Adopt delay as congestion signal. Delay is in the form of RTT measurements. RTT is a fine-grained measure of congestion that comes with **every ACK**. It effectively supports multiple traffic classes by providing an inflated measure for lower-priority transfers that wait behind higher-priority ones. Further, it requires no support from network switches.

> Why every ACK? -- RTT's limitation.
> RTT measurements lump queueing in both directions along the network path. This may confuse reverse path congestion experienced by ACKs with forward path congestion experienced by data packets. One simple fix is to send ACKs with higher priority, so that they do not incur significant queuing delay.

- Unlike earlier schemes, TIMELY does not build the queue to a fixed RTT threshold. Instead, it uses **the rate of RTT variation**, or the gradient, to predict the onset of congestion and hence keep the delays low while delivering high throughput. By using the gradient, we can react to queue growth without waiting for a standing queue to form – a strategy that helps us achieve low latencies.

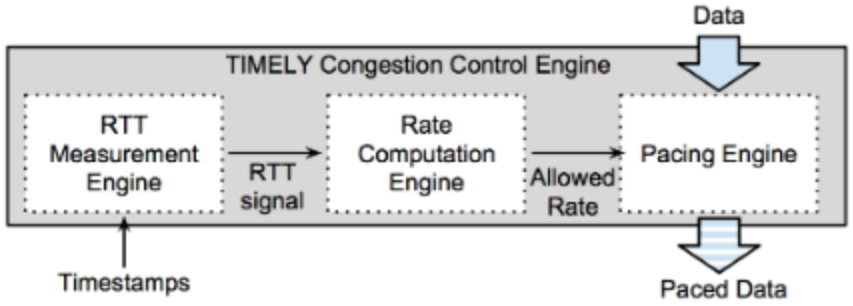


Figure 6: TIMELY overview.

- RTT Measurement: $RTT = \mathrm{t}_{\mathrm{completion}} - \mathrm{t}_{\mathrm{send}} - \frac{\mathrm{seg.size}}{\mathrm{NIClinerate}}$,
- Rate Computation:
  - Computing the delay gradient
  - Computing the sending rate (AI/MD): $R = R + \delta, R = R\left(1 - \beta d \frac{RTT(t)}{dt}\right)$
- Pacing: TIMELY is rate-based rather than window-based because it gives better control over traffic bursts given the widespread use of NIC offload. Windows do not provide fine-grained control over packet transmissions. It is easier to directly control the gap between bursts by specifying a target rate.

Swift

Overview

A improved version of TIMELY by Google.

Innovation points:

By **using the difference between the RTT and target delay rather than the RTT gradient**, Swift is more simple than TIMELY.

To mitigate staleness concerns in using delay as a congestion signal:

- use **instantaneous delay** as opposed to minimum or low-pass filtered delay.
- do not explicitly delay ACKs.

Advantages:

- 使用延迟作为拥塞信号，因而部署简单，能够适应快速变化的数据中心的技术
- 像 TIMELY 一样高效利用 CPU 和 NIC 资源，保证对 CPU 的低占用率
- 有效处理 large-scale incast 等流量模式(通过设置 cwnd<1)，即使在 $\mathcal{O}(10k)$ 规模的流量情况下也能保持高 IOPS
- 将延迟分解为主机延迟（host delay）和结构延迟（fabric delay），分别对不同的拥塞原因进行测量、响应、优化
- 在不同工作负载的集群中都能维持较低的排队延迟水平，提供较高的利用率，并且保证接近零的丢包率

HPCC

Overview

HPCC leverages in-network telemetry (INT) to obtain precise link load information and controls traffic precisely. By addressing challenges such as delayed INT information during congestion and overreaction to INT information, HPCC can quickly converge to utilize free bandwidth while avoiding congestion, and can maintain near-zero in-network queues for ultra-low latency. HPCC is also fair and easy to deploy in hardware.

Innovation points:

INT
precise information based MIMD

Cebinae