RDMA Congestion Control: ECN or Delay?

Vishal Misra  
Columbia University, joint work with  
Yibo Zhu, Monia Ghobadi, Jitendra Padhye (all Microsoft)
Outline

• Why RDMA Congestion Control?
• Congestion Signals
  • Google approach: delay
  • Microsoft approach: ECN
• ECN vs Delay comparison
  • Stability, Speed of Convergence
  • Fixed points
  • Flow completion time comparison for a standard datacenter benchmark
  • ECN: faster feedback. Delay: slower, distorted feedback
  • PI control: fundamental tradeoff
Host TCP Stack is Heavyweight

40Gbps NICs, state-of-the-art servers, 16 cores

Small messages → CPU is the bottleneck

Larger msgs → ~3 CPU cores are burnt by TCP

Sender

Receiver

Throughput (Gbps)

CPU utilization (%)

Message size

TCP

RDMA (read/write)

Time to transfer 2KB (ms)

Message size

TCP

RDMA (send)

CPU utilization (%)

Message size

TCP
Solution: RDMA

RDMA bypasses host OS stack ➔ frees host CPU, lowers latency
RDMA Outperforms TCP

RDMA single thread ~40Gbps

RDMA CPU ~0%
RDMA in Modern Datacenters

- In past, RDMA deployed on special fabrics, i.e., InfiniBand
- InfiniBand incompatible with Ethernet + IP
- **Solution**: RoCEv2 (RDMA over Converged Ethernet)
- **Problem**: RoCEv2 has very blunt congestion control called “PFC”
  - Stop flows when queues build up
Enter DCQCN and TIMELY: Congestion Control for ROCEv2

DCQCN (Microsoft)
- Based on DCTCP
- Switch marks packets on detecting congestion (ECN)
- Receiver reflects marked packets via ACKs
- Sender adjusts rate using DCQCN algorithm
- Ongoing deployment on Microsoft Azure

TIMELY (Google)
- Based on TCP Vegas
- Switch plays no role (FIFO queue assumed)
- Receiver sends ACKs (once per burst)
- Sender estimates Delay, and responds to derivative.
- Ongoing deployment at Google
Two solutions to the same problem

- **Key difference: ECN vs. Delay**
  - There are other differences as well – e.g. hardware packet pacing

- **Comparing their design and performance can yield valuable insights**

- **Properties we care about:**
  - Stability: flow rates and queue length stabilize
  - Fast convergence: system should stabilize quickly
  - Fairness: at stable point, flows should share bandwidth equally
  - High link utilization
Methodology

- **Fluid model**
  - For analytical results

- **... backed by NS simulations**
  - For packet-level results

- **.... backed by (in case of DCQCN) implementation comparison**
  - to ensure some connection to reality

- **Assumptions**
  - Long lived flows
  - Identical RTT
  - Single shared bottleneck
Some equations to impress you ...

\[ p(t) = \begin{cases} 0, & q(t) < K_{\min} \\ \frac{q(t) - K_{\min}}{K_{\max} - K_{\min}}, & K_{\min} < q(t) \leq K_{\max} \\ 1, & q(t) > K_{\max} \end{cases} \]  

\[ \frac{dq}{dt} = \sum_{i=1}^{N} R_{i}^{(i)}(t) - C \]  

\[ \frac{dR_{i}}{dt} = \begin{cases} \frac{g_{i}}{\tau_{i}}, & q(t - \tau') < C \times T_{\text{low}} \\ -\frac{g_{i}}{\tau_{i}}, & q_{i} \leq 0 \\ -\frac{1}{\tau_{i}} \left( 1 - \frac{C \times T_{\text{high}}}{q(t - \tau')} \right) R_{i}(t), & g_{i} > 0 \end{cases} \] 

\[ \frac{dg_{i}}{dt} = \frac{\alpha \left( -g_{i}(t) + \frac{q(t - \tau') - q(t - \tau' - \tau_{i}^*)}{C \times R_{i} \times D_{\text{min RTT}}} \right)}{\tau_{i}^*} \] 

\[ \tau_{i}^* = \max \left\{ \frac{\text{Seg}}{R_{i}}, D_{\text{min RTT}} \right\} \] 

\[ \tau' = \frac{q}{C} + \frac{MTU}{C} + D_{\text{prop}} \] 

Figure 7: TIMELY fluid model

Takeaway: DCQCN is a little too complicated
... those equations do model “reality”

DCQCN model matches simulations and implementation

TIMELY model matches simulations
Congestion Control: Desirable properties

- **Stability**
  - Queue does not oscillate (or worse, exhibits runaway behavior)

- **Rate of convergence**
  - Quickly converge to stable operating point

- **Fairness**
  - At convergence, all flows equally share bottleneck bandwidth

- **High utilization**
  - Otherwise, you can achieve all of the above by dropping all packets

- **Low flow completion time**
  - But without doing fancy stuff at the switch
• DCQCN has a unique fixed point
• At the fixed point, all flows share the bottleneck equally
• Convergence is fairly rapid
• Relationship between stability and number of flows is non-monotonic
TIMELY

• Timely has no fixed point
  • changes rate in response to changes in latency (derivative)
  • Can stabilize at any point where sum of rates = bottleneck bandwidth

(a) Both flows start at time 0 at 5Gbps  (b) Both start at 5Gbps, one starts 10ms late  (c) One starts at 7Gbps, the other at 3Gbps
“Web Search” workload experiments
Why is TIMELY performing poorly?

- Reliance on delay differential
  - Can be fixed by making rate changes in response to absolute delay
- Feedback is delayed as queue builds up
- Can have fixed queue or fairness – but not both!
- ECN marking is resistant to feedback jitter
Why is TIMELY performing poorly?

- Reliance on delay differential
  - Can be fixed by making rate changes in response to absolute delay
- Feedback is delayed as queue builds up
- Can have fixed queue or fairness – but not both!
- ECN marking is resistant to feedback jitter
What happens with ECN

- **T0, Q = 2**: Blue packet is about to arrive
- **T1, Q = 3**: Blue packet arrival complete
- **T2, Q = 4**: Blue packet ready to depart... and is marked, reflecting state of queue at T2

Marking threshold = 4 packets
What happens with delay

**T0, Q = 2**

Blue packet is about to arrive

**T1, Q = 3**

Blue packet arrival complete. ... timer starts

**T2, Q = 4**

Blue packet ready to depart ... and reflects state of queue at T0
In other words

- Delay inherently reports "stale" information
- The staleness is affected by queue length!
  - Longer queue $\Rightarrow$ more stale feedback
- This can lead to instability
Why is TIMELY performing poorly?

- Reliance on delay differential
  - Can be fixed by making rate changes in response to absolute delay

- Feedback is delayed as queue builds up

- Can have fixed queue or fairness – but not both!

- ECN marking is resistant to feedback jitter
A problem with both DCQCN and TIMELY

DCQCN (40Gbps link)

TIMELY (10Gbps link)
Converge to a fixed queue length regardless of number of flows

- FCT is more predictable!

- Can be done with a Proportional-Integral (PI) controller

- Cisco’s variant of PI (PIE) part of DOCSIS 3.1 standard to control bufferbloat in consumer cable modems

- DCQCN → use PI controller to mark packets
  - instead of RED-like marking

- TIMELY → implement PI controller at the host with delay as the signal
PI controller works with DCQCN

DCQCN with RED-like marking

DCQCN with PI-like marking
PI Controller with TIMELY: lose fairness

- 2 Flows
- 10 Flows
Fundamental limitation

- Delay-based protocols can have fixed queue or fairness – but not both!

- Proof sketch:
  - N flows need to make decisions separately (i.e. distributed), and calculate C/N to be their fair share
  - At steady state, since delay is fixed, this feedback is independent of the number of flows.
  - Need an additional variable to signal "N" back to the flows, and that is the ECN marking probability
Why is TIMELY performing poorly?

- Reliance on delay differential
  - Can be fixed by making rate changes in response to absolute delay

- Feedback is delayed as queue builds up

- Can have fixed queue or fairness – but not both!

- ECN marking is resistant to feedback jitter
Impact of reverse path delay

ECN is more resistant as feedback signal is only delayed.
With Delay, the feedback signal is both delayed and distorted.
Analogy: Decoupling Signal from Noise

NOISE IN AM & FM SYSTEMS

- Noise from other sources is AM noise.
- FSK Waveform.
- FSK Waveform with noise.

Delay
ECN
Conclusion: ECN appears better

- Generally stable
- Fair & converging
- Delay increases with the number of flows
- Sensitive to variable feedback delay

- Generally stable
- Addressed bufferbloat
- Fixed delay regardless of the number of flows
- Sensitive to variable feedback delay

- Generally stable
- Fair & converging
- Addressed bufferbloat
- Resilient to variable feedback delay

- Generally stable
- Addressed bufferbloat
- Resilient to variable feedback delay

(TIMELY)

(Patched)