TIMELY: RTT-based Congestion Control for the Datacenter

Authors: Radhika Mittal(UC Berkeley), Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi(Microsoft), Amin Vahdat, Yaogong Wang, David Wetherall, David Zats

Presenters: Buting Ma, Xinghao Li

Outline

1. Multi-bit RTT signals measured with NIC hardware are strongly correlated with network queueing.

2. Transport Informed by MEasurement of LatencY (TIMELY): an RTT-based congestion control scheme.

3. Evaluation of TIMELY with an OS-bypass messaging implementation using hundreds of machines on a Clos network topology.

RTT can be measured accurately

IMPORTANT: Network Interface Controller (NIC, hardware) support

1. Time is logged by hardware; 2. ACK is done by hardware

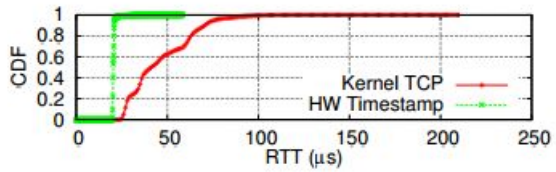


Figure 1: RTTs measured by hardware timestamps have a much smaller random variance than that by kernel TCP stack.

Advantages of using RTT

Need no support from switch (*but support from NIC)

RTT directly reflects latency, but Explicit Congestion Notification (ECN) only marks queue length threshold

Accumulate information about end to end path, ECN only reflects a single switch

RTT contains multiple bits information (fine grained, gradient possible), ECN is only binary

*RTT not suitable for large area network due to paths with various length;

In datacenter, all paths have propagation delays, and in Clos topology, same paths have same length, a measurable constant

RTT closely related to queue occupancy

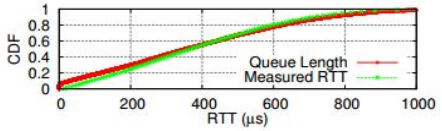


Figure 2: RTTs measured at end-system track closely the queue occupancy at congested link.

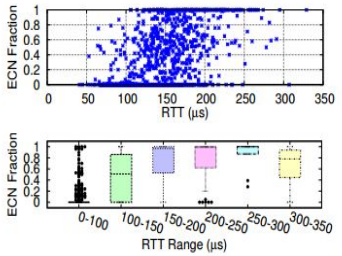


Figure 3: Fraction of packets with ECN marks versus RTTs shown as scatter plot (top) and box plot (bottom).

Strong correlation between RTT and queue length

Only weak correlation between ECN marks and RTT

Problem with reverse path congestion

confuse reverse path congestion experienced by ACKs with forward path congestion experienced by data packets.

Solution: send ACKs with higher priority

(No need for more complicated methods)

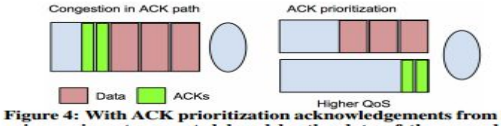


Figure 4: With ACK prioritization acknowledgements from the primary incast are not delayed by the data of the secondary incast.

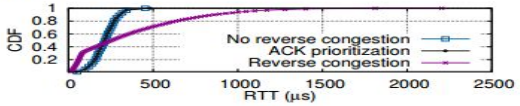


Figure 5: In the presence of reverse congestion, RTT measurements with ACK prioritization are indistinguishable from RTTs that do not experience any reverse path congestion.

TIMELY: Transport Informed by MEasurement of LatencY

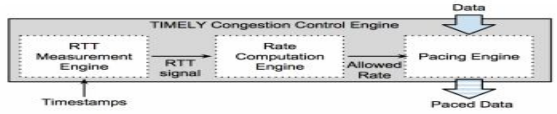


Figure 6: TIMELY overview.

1) RTT measurement to monitor the network for congestion;

2) a computation engine that converts RTT signals into target sending rates; and

3) a control engine that inserts delays between segments to achieve the target rate.

*Independent instance for each flow

RTT measurement

1) the serialization delay to transmit all packets in the segment, typically up to 64 KB; (not included)

2) the round-trip wire delay for the segment and its ACK to propagate across the datacenter; (small & constant)

3) the turnaround time at the receiver to generate the ACK; (negligible)

4) the queuing delay at switches experienced in both directions.

$$RTT = t_{completion} - t_{send} - \frac{\text{seg. size}}{\text{NIC line rate}}$$

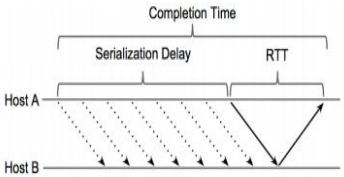


Figure 7: Finding RTT from completion time.

Congestion control algorithm

Algorithm 1: TIMELY congestion control.

Data: new_rtt
Result: Enforced rate
new_rtt_diff = new_rtt - prev_rtt ;
prev_rtt = new_rtt ;
rtt_diff = (1 - α) · rtt_diff + α · new_rtt_diff ;
▷ α: EWMA weight parameter
normalized_gradient = rtt_diff / minRTT ;
if *new_rtt* < T_{low} **then**
  rate ← rate + δ ;
  ▷ δ: additive increment step
  return;
if *new_rtt* > T_{high} **then** exponentially
  rate ← rate · $\left(1 - \beta \cdot \left(1 - \frac{T_{high}}{new\_rtt}\right)\right)$;
  ▷ β: multiplicative decrement factor
  return;
if *normalized_gradient* ≤ 0 **then**
  rate ← rate + N · δ ;
  ▷ N = 5 if gradient<0 for five completion events (HAI mode); otherwise N = 1
else
  rate ← rate · (1 - β · normalized_gradient)

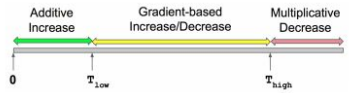


Figure 8: Gradient tracking zone with low and high RTT thresholds.

proportional–integral–derivative control (PID control) without integral

T_low: no empty queue; T_high: no long queue

Empty queue: low latency and low throughput; long queue: high latency and high throughput

Exponentially Weighted Moving Average (EWMA) filter: detect the overall trend in the rise and fall in the queue, while ignoring minor queue fluctuations that are not indicative of congestion

Hyperactive increase (HAI) for faster convergence

Gradient approach vs. Queue size approach

Set T_high = T_low =: T_target

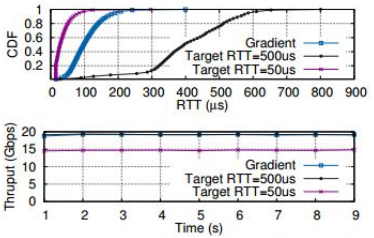


Figure 9: Comparison of gradient (low and high thresholds of 50 μs and 500 μs) with target-based approach (T_{target} of 50 μs and 500 μs).

Both low latency and high throughput

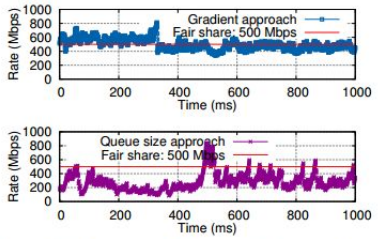


Figure 10: Per-connection rates in the gradient approach are smooth (top) while those in the queue-size based approach (with T_{target} = 50μs) are more oscillatory (bottom).

More smooth traffic and better use of share

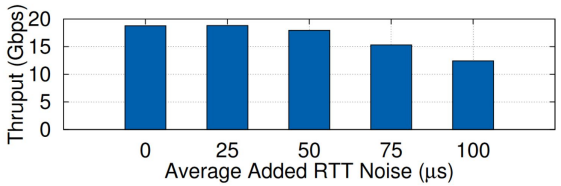
Evaluations

- Evaluate the TIMELY at two scales:
  1. A small-scale testbed (a single rack).
     - Throughput
     - Fairness
     - Packet latency
     - Timing accuracy
  2. Hundreds of machines in a classic Clos network topology.
     - Traffic workload
     - Hosts collect measurements of per-connection throughputs
     - RPC latencies
     - RTTs

1. Small-Scale Experiments

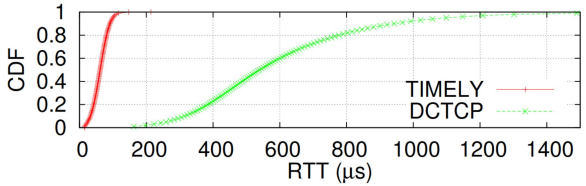
The accuracy of RTT samples

- Observe the impact to the throughput from added RTT noise
  - Uniformly distributed RTT noise from 0 to n (x-axis).



- Higher noise leads to more throughput degeneration. (expected behavior)

TIMELY vs DCTCP - RTT



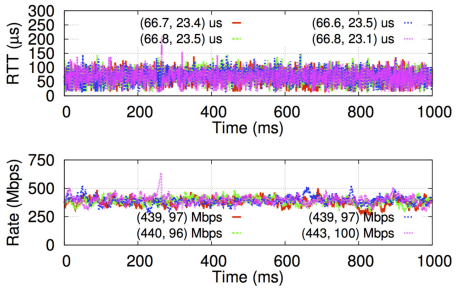
- TIMELY keeps average RTT 90% lower than that of DCTCP
- TIMELY keeps tail RTT 92% lower than that of DCTCP
- No throughput reduction for TIMELY

Performance Comparison

Metric	DCTCP	FAST*			PFC	TIMELY
		10M	50M	100M		
Total Throughput (Gbps)	19.5	7.5	12.5	17.5	19.5	19.4
Avg. RTT (us)	598	19	120	354	658	61
99-percentile RTT (us)	1490	49	280	460	1036	116

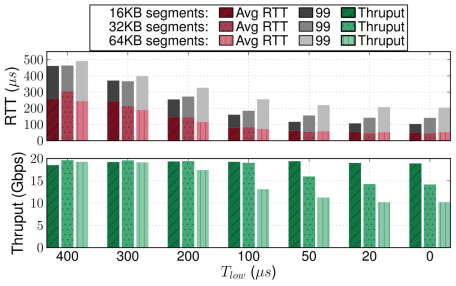
- TIMELY has about 90% shorter RTT than DCTCP and PFC
- FAST has better RTT with low throughput. But the RTT is still more than 5x longer with higher bandwidth that is comparable to TIMELY

TIMELY - Fairness



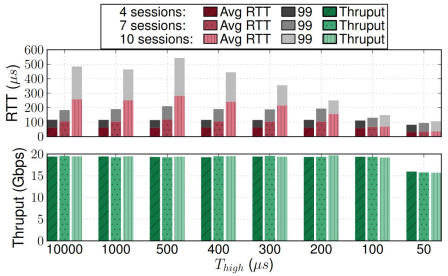
- The throughput is close to the fair share (4 connections, 500Mbps each).
- The RTT remains low.

Throughput and RTT vs different T_{low}



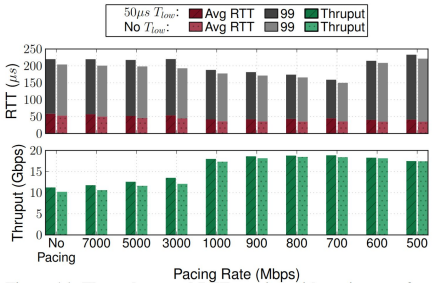
- Shorter T_{low} leads to shorter RTT
- Larger segments leads lower throughput with shorter T_{low}
- It is hard for large segments to maintain both short RTT and high throughput

Throughput and RTT vs different T_{high}



- More sessions leads to higher RTT
- Shorter T_{high} may reduce the throughput
- Optimal T_{high} is between 100 and 200 microseconds

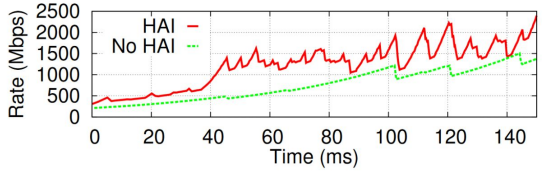
Throughput and RTT vs different pacing rate



- Smoothes the bursts
- Segment size is 64KB
- Greater pacing (lower pacing rate) leads to shorter RTT and higher throughput
- Optimum pacing rate is 700Mbps

Hyper Active Increment (HAI)

- Initial fair rate = 200 Mbps (10 connections)
- Target fair rate = 2000 Mbps (1 connection)

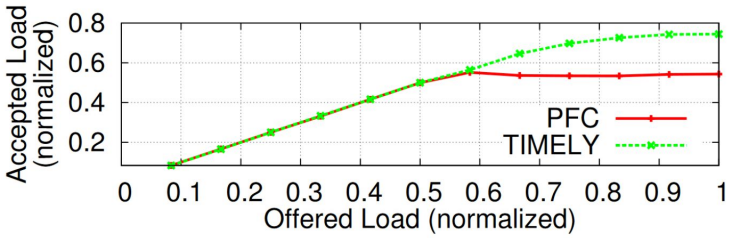


- HAI makes TIMELY to reach the target rate faster.

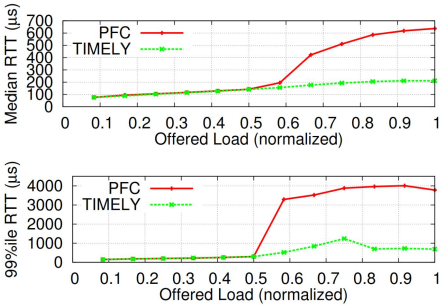
2. Large-Scale Experiments

TIMELY vs PFC - Saturated Load

- TIMELY has a higher saturated load than that of PFC

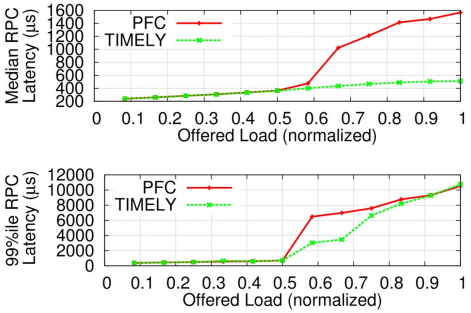


TIMELY vs PFC - RTT (with RPC requests)



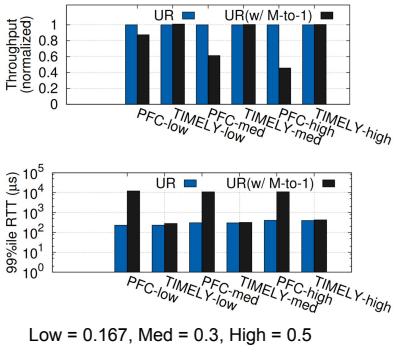
- Each client pick up a server with the longest path
- Uniformly and randomly send 64KB RPC (remote procedure call)
- TIMELY leads to 60% shorter median RTT and 75% shorter 99 percentile RTT

TIMELY vs PFC - RPC Latency



- 70% shorter median RPC latency for TIMELY (with high load)
- Same 99 percentile RPC latency for both TIMELY and PFC (because end-host queueing delay is included in RPC latency)

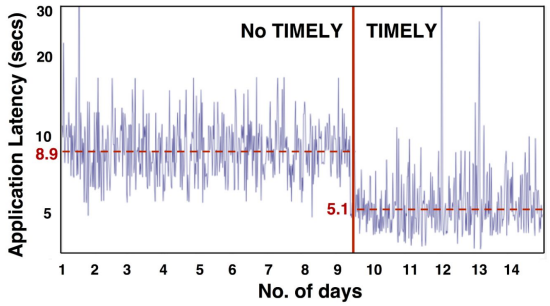
Network imbalance



Low = 0.167, Med = 0.3, High = 0.5

- Uniform random background traffic + added incast load
- For PFC, with added incast load, the throughput reduced significantly. And the RTT increased about 100%
- For TIMELY, both throughput and RTT does not have noticeable change with added incast load

Application Level Benchmark (Storage)



Summary

- RTT correlates well with queue buildups and congestion level in data centers
- TIMELY is able to respond to microsecond-level RTT change (with NIC support) and adjust the data rate to effectively mitigate the congestion and increase the overall throughput while reduce the RTT

Discussion

- What is one queue decreasing but another one increasing?
- Including integral part of PID, helpful for historic fairness?
- How about building the TIMELY into NIC hardware (chip)?
- New algorithm (e.g. machine learning based) to dynamically tune the parameters in the algorithm.
- Will TIMELY work well in the Internet?