Moving Beyond Proxy Signals for Datacenter Congestion Control

CS 244 – Final Project Report

Serhat Arslan
Dept. of Electrical Engineering, Stanford University
sarslan@stanford.edu

Catalin Voss
Dept. of Computer Science, Stanford University
catalin@cs.stanford.edu

ABSTRACT

Efficient congestion control in datacenters remains a challenge. Multiple approaches have recently been proposed to move beyond packet loss as the central proxy signal for congestion in the datacenter setting. Google developed TIMELY, which uses precise RTT measures obtained at the NIC as the proxy signal for congestion. DCTCP and DCQCN have argued for using ECN bit as the proxy instead. We aim to contribute to this controversy in two parts. First, we attempt to reproduce the original TIMELY results in an NS-2 simulation. Our reproduction covers the small scale experiments presented by the authors. We run the experiments with the same topology and obtain Figure 13 and 14 of the original paper. While we are indeed able to reproduce RTT results that are similar to the TIMELY claims, we observed that the DCTCP implementation in our simulations performed better than the baseline used in the TIMELY paper. We also find that TIMELY flows do not converge to the fair rate in our simulation and that TIMELY may trade throughput for lower latency in larger topologies.

We then conjecture that this is because RTT is a noisy signal, containing the signal from the bottleneck (important), plus signals from all the lightly congested non-bottleneck queues (not so important). To investigate this issue and reconcile the debate between ECN and delay-based schemes, we explicitly test the impact of hop-by-hop queuing delay on the congestion algorithm. We infer that the sum of these signals – RTT – is a noisy approximation for congestion. Other functions of hop-by-hop queue occupancy may serve as more accurate congestion signals and may present an avenue for further research to develop multi-bit ECN schemes.

CS 244 Final Project, 2019, Stanford University

1 INTRODUCTION

Congestion control remains a significant challenge for today's demanding web applications, which simultaneously require high throughput and low latency. Even in modern datacenters, where we have complete control over the network topology and end-hosts, there remains much debate about what the right congestion signal might be. TCP Tahoe and Reno use packet loss as their feedback mechanism whereas DCTCP [1] and Microsoft's DCQCN [10] use ECN. Google's TIMELY [8] argues that a better proxy signal for congestion could be precise, microsecond-level RTT measurements taken at the NIC. We aim to contribute to this controversy in two parts.

First, in section 2, we attempt to reproduce the original TIMELY results in an NS-2 [5] simulation. We seek to recreate figures 13 and 14 from the TIMELY paper [8], showing comparison of performances between TIMELY and DCTCP. Since the TIMELY testbed has not been open-sourced, we re-implemented the scheme with the help of a code snippet [7] published by an author of the paper. We also added noise perturbation to simulate real RTT measurements. We conclude that TIMELY's original RTT measurements hold up to reproduction. However, our reference implementation of DCTCP significantly outperforms Google's, leading to much smaller marginal gains. Further, we observe occasional unfair flow allocations when we repeatedly investigate four random flows that were chosen for illustration in figure 13 of [8]. Zhu et al. argue that this behavior can be fixed by introducing a stable point in TIMELY [12]. We re-implemented their "Patched TIMELY" algorithm and find that it may resolve the problem at the cost of some additional variance that has to be tuned manually. Indeed, even TIMELY's authors confirmed to us that they have moved away from the gradient-based algorithm since the publication of the original TIMELY paper.

Next, in section 3, we consider the performance of TIMELY in larger topologies. Motivated by Zhu et al.'s criticism of TIMELY in [12], we seek to test how TIMELY is affected by the presence of non-bottleneck queues in the network. We hypothesize that when non-bottleneck links oscillate, RTT naturally becomes a more noisy signal. To test TIMELY's behavior in this setting, we perform experiments on a long

skinny topology and a larger fat tree topology. We observe that longer TIMELY flows go below their fair share. The results from our ablations suggest that TIMELY may trade latency for throughput in larger topologies. In section 3.2, we explore alternative ways of addressing the RTT noise problem. We point out that TIMELY, just like its ECN-based colleagues DCTCP and DCQCN, acts upon some (perhaps noisy) function of the hop-by-hop round-trip queue occupancy vector. For a path with k hops, we can write this vector as $q = (q_1, ..., q_{2k})$. Arguably, q contains the root congestion signal. We argue that q can easily be obtained in the datacenter setting via programmable switches as demonstrated by INT [6] and other approaches. Then it remains to write the congestion signal for TIMELY and other schemes as a function $f(q)$. We experiment with multiple functions $f(\cdot)$. When we use $f(q) = \sum_{i=1}^{2k} q_i = RTT$ as our congestion signal, we get TIMELY. When we consider $f(q) = \mathbb{I}\{\exists_i q_i > thresh\}$, we get an ECN-based scheme. We test how functions that include more of the bottleneck signal and less of the oscillating non-bottleneck signals compare to the TIMELY formulation and find that $f(q) = \max_i q_i$ mitigates the fairness issue and results in lower flow completion times although it might still be a non-optimal one.

We briefly discuss how such an approach could be realized as a multi-bit ECN scheme via programmable switches, e.g. using INT [6] in section 4. Our discussion does not attempt to fully solve the congestion control problem, but motivates the use of non-proxy, nanosecond-level accurate signals for improving our understanding of communication networks. We leave additional experiments for future work.

2 TIMELY REPRODUCTION

2.1 Background

In contrast to previous delay-based schemes such as TCP Vegas, Google's "Transport Informed by MEasurement of LatencY" – TIMELY – obtains its RTT estimates through hardware timestamps. Just before posting a packet segment to the NIC, the OS queries the NIC for its current hardware time and records this as t_{send}. The NIC then provides the hardware time for the first ACK in the segment back to the OS as $t_{\text{completion}}$. From this, the OS computes an RTT that is reduced to estimate the queueing delay:

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

The TIMELY algorithm then acts on the delay-gradient computed using this signal, to update the sending rate as specified in Algorithm 1 given in [8]. The main intuition behind this algorithm is to slow down the sending rate by an amount that depends directly on how much congestion rises. If RTT is less than a threshold T_{low}, the rate is increased additively by some δ. If, on the other hand, $RTT > T_{\text{high}}$, the rate is decreased multiplicatively by some β. If it RTT is in between T_{low} and T_{high}, the rate is updated using the delay gradient.

2.2 Previous Critiques of TIMELY

The concurrent release of TIMELY by Google and DCQCN by Microsoft caused a fundamental debate between using delay vs. ECN as a congestion signal. This has been the topic of several papers, with some arguing that we ought to combine the two [9]. Microsoft's initial critique of TIMELY by Zhu et al. used an NS-3 simulation and fluid models to argue that TIMELY lacks a fixed point [12]. The authors argue that, as a result, TIMELY flows are not guaranteed to converge to the fair port and the system is generally unstable. They propose a "patched" version of Algorithm 1, where the rate decrease is modified to depend on absolute RTT rather than the RTT gradient and a gradient dependent weight function is used to smoothen the transition between rate increase and rate decrease and avoid harsh oscillations. Indeed, we have confirmed with the Timely authors that using only gradient caused some flows to achieve higher/lower rates than the fair share rate. In later versions of Timely, they have also modified the algorithm to start using target RTT thresholds instead of gradient.

Algorithm 1: TIMELY congestion control

Data: new_rtt
Result: Enforced rate
new_rtt_diff = new_rtt - prev_rtt ;
prev_rtt = new_rtt ;
rtt_diff = $(1-\alpha)\cdot$ rtt_diff + $\alpha\cdot$ 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**
  rate ⟵ rate $\cdot (1 - \beta \cdot (1 - \frac{T_{high}}{new\_rtt}))$;
  ▷ β: multiplicative decrement factor
  return ;
if $normalized\_gradient \leq 0$ **then**
  rate ⟵ rate + N $\cdot\delta$;
  ▷ N=5 if gradient<0 for 5 completion events (HAI mode); otherwise N=1
else
  rate ⟵ rate $\cdot (1 - \beta\cdot$ normalized_gradient)

We attempt to construct a scenario that showcases this issue in section 3. We also consider the fixed-point TIMELY implementation in our ablation experiments. Because "Patched TIMELY" includes an additional hyper-parameter RTT_{ref} to scale the rate decrease, which isn't described in [12] and which may be tuned for the topology, we caution about generalizing these results beyond the scope of our reproduction.

Motivated by the ECN vs. delay controversy and the trend towards combining the end-to-end and hop-by-hop signals, we investigate the breakdown of the signal used by TIMELY in section 3.2.

2.3 Reproduction Overview

We build our reproduction on the DCTCP implementation patch for NS-2 given in [3]. We implement TIMELY in C++ by adapting the default TCP Vegas implementation in NS-2. We use a C++ snippet provided by one of the authors [7] as the basis for our implementation of the TIMELY algorithm. Because the TCP implementation in NS-2 requires a congestion window and cannot be directly controlled using a rate, we convert the output rate back to `cwnd` using the classical relation `cwnd` = $RTT \times$ `rate`. This has the unavoidable side effect that the rate is effectively discretized as `cwnd` cannot enforce a fractional number of packets in flight. The effects of this phenomenon can be seen in the step function behavior in our results. However, this does seem not significantly change the overall measurements, i.e. RTT values are still bounded by the continuous results of the original TIMELY paper.

2.4 Small-Scale Reproduction

We first reproduce TIMELY's small-scale experiments in the original topology implemented in NS-2. We use 10 clients and one server, with 4 connections per client. We emulate an incast scenario using long-running FTP flows. The TIMELY experiments are done with 2×10G uplinks. Since NS-2 doesn't provide any traffic balancing, we instead use a single 20G uplink per client. This may impact the packetization delay slightly.

To simulate the noise associated with measuring the RTT, the TIMELY evaluation adds random, uniform noise $\in [0, x]\mu s$, where x is set to some number between 0 and 200. The noise value used is not specified for all plots. Since TIMELY obtains its measurements at the NIC and the authors argue that performance decays significantly with $x \geq 50$, we perform our primary experiments with $x = 25$.

In Figure 1, we show the CDF of the RTT distribution for TIMELY and DCTCP in the small-scale experiment setup across all flows. The CDF for our TIMELY simulation seems to have comparably similar head and tail distributions with

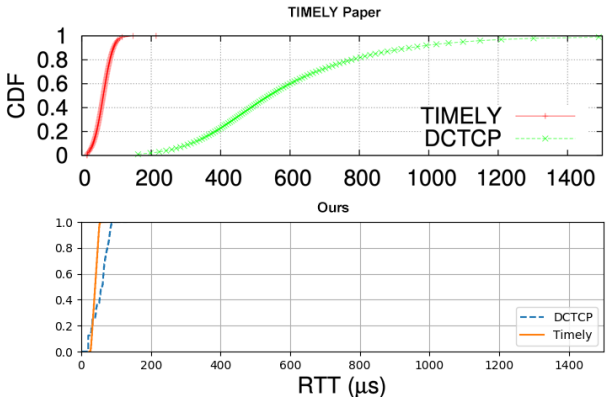

Figure 1: CDF of RTT distribution comparing DCTCP and TIMELY in (Figure 14 in the TIMELY paper [8]) using 25μs of RTT noise. A curve that is further on the left means that protocol achieved lower latency in general. The original reference figure is given above our simulation results. The results for TIMELY are comparable, but our reference DCTCP implementation performs significantly better than the baseline used in [8]. With more than 50μs of measurement noise, DCTCP begins to outperform TIMELY.

[8]. Its straight line form may be due to the discretization behavior of NS-2 as discussed in section 2.3. Our results match the shape of the original paper, but the gap between our DCTCP implementation and TIMELY is much lower than asserted by the TIMELY paper. Our DCTCP implementation relies on the NS-2 patch [3] published by the authors of DCTCP and we have used the default parameters used by a reproduction project that was completed at Stanford University [4]. [8] presents a limitation for their implementation of DCTCP that they run the algorithm in an optimized kernel without PFC support whereas Timely is implemented with OS-bypass messaging. The reason for DCTCP to not run with OS-bypass messaging is given as NIC firmware limitations on processing ECN feedback [11]. We have communicated this issue with the authors and learned that the main contribution to high latency with DCTCP came from the queuing created at the end-host NICs that DCTCP has no ability to react to. NS-2 does not provide NIC abstraction which prevented us to observe such queuing with DCTCP. All the queues in our simulations have ECN markings by nature which makes DCTCP react.

In figure 2, we plot the RTT achieved by TIMELY across time for four randomly chosen flows, reproducing the first part of Figure 13 from the TIMELY paper. We observe a mean RTT of 40μs (SD=7μs) vs. the mean of 66.8μs (SD=23.5μs) reported in the TIMELY paper. The difference between the

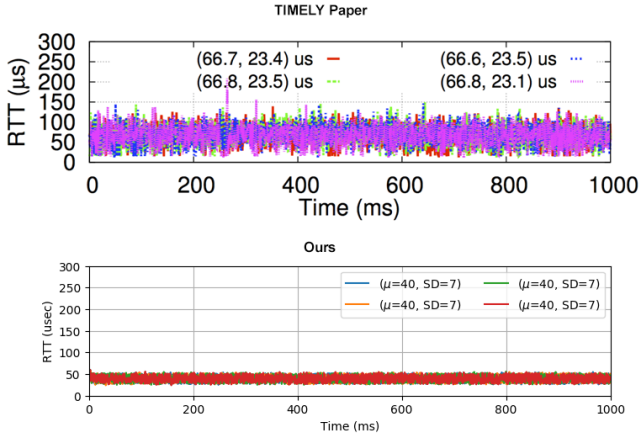

Figure 2: TIMELY RTT for four randomly selected flows across time, corresponding to Figure 13 in the TIMELY paper [8]. Mean RTT and the standard deviation are given for each flow in parentheses. At a noise level of 25μs, we observe a lower mean RTT, but note that our simulation may exhibit lower packetization delay.

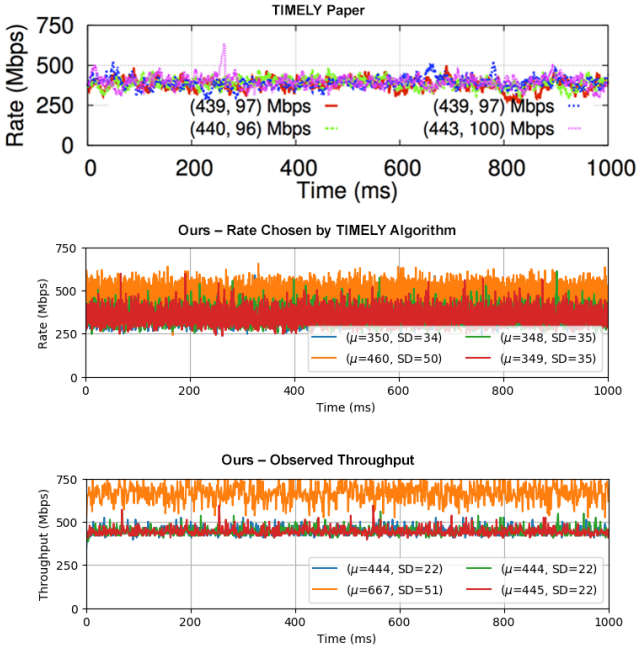

Figure 3: TIMELY sending sending rate and throughput, corresponding to Figure 13 in the TIMELY paper [8]. The authors only report a single rate. We investigate both the sending rate chosen by TIMELY (middle) and the actual observed throughput (bottom). Mean Mbps and the standard deviation are given for each flow in parentheses. [8] states that the top figure was generated by choosing four random flows. We ran this procedure several times to identify the flows below, which do not all share the same rate. The same four flows are included in the middle and bottom figures.

RTT measures may be explained with the lower packetization delay in our simulation and the choice of noise. Like in the TIMELY paper, we observe that the mean RTT increases rapidly as we increase the measurement noise past 50μs (not depicted).

In Figure 3, we reproduce the rate plots from Figure 13 of the TIMELY paper. In the original paper, these plots served to show that TIMELY results in fair allocation at the proportionally fair share of 500 Mbps in this topology. The TIMELY paper plots only the rate as the actual throughput achieved. We also plot the rate chosen by the algorithm. The authors claim to have chosen four random flows. We also choose four random flows, but we repeat this randomization several times and observe very different graphs. In some cases, all of the flows exhibit the same rate, like in the TIMELY paper. However, in some cases, as depicted here, we observed that TIMELY assigns consistently higher rates to some flows, even in this simple topology. We further observe that while the assigned rate difference is relatively small, this difference is accentuated once we look at the observed throughput. Stability and fairness of the algorithm is analyzed by [12] as we also discussed in section 2.2.

We repeat the experiment with [12]'s patched TIMELY algorithm. The results are given in figure 4. We found the rates chosen by patched TIMELY depended significantly on the choice of the hyper-parameter RTT_{ref}. Since RTTs are on the order of 30μs, we choose $RTT_{ref} = 50\mu$s as a reasonable parameter to achieve positive queues. With this setting, we find that Patched TIMELY indeed appears to remedy the flow fairness issue but also significantly increases variance. We have begun exploring the choice of RTT_{ref} with the authors of [12] but leave a detailed investigation for future work.

3 ABLATION TOPOLOGIES

We now turn to testing TIMELY's performance in larger topologies with additional non-bottleneck queues.

3.1 Skinny Topology

We first construct a skinny topology shown in figure 5. A primary multi-hop connection from source to destination has to cross several secondary cutting flows. The black arrows on the figure indicate the secondary flows that are continuously active throughout the simulation in order to generate queuing on the links. The last two links of the path are congested

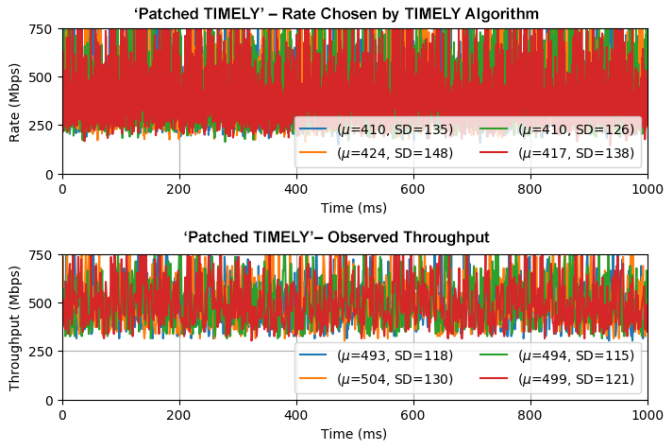

Figure 4: Patched TIMELY results (compare to figure 3). We find that Patched TIMELY indeed appears to remedy the flow fairness issue but significantly increases variance. We caution that more testing with the hyperparameter RTT_{ref} (set to 50μs here) would have to be performed to draw general conclusions from this.

relatively more as there are more flows to share the queues. All links are set to 10Gbps. All hosts use the same tested congestion signal (described further in section 3.2). To ensure that the "side flows" caused oscillating non-bottleneck queues along the primary path significantly and fairly, the cross flows' link delays are set such that all flows in the topology have the same base-RTT (delay with no queuing). This ensures that the flows change their rate equally in response to the same amount of change in delay. We investigate the behavior of the single benchmark connection flow between the nodes that are indicated as source and destination.

During the simulation, all flows fill up buffers until they reach T_{low} amount of delay. Side-flows across non-bottleneck links tend to see a large positive RTT gradient after this point and thus decrease their rates, causing the queue occupancy in the corresponding links to decrease. On the other hand, the links with multiple cross-flows tend to have a relatively more stable queue occupancy since the rates of flows sharing

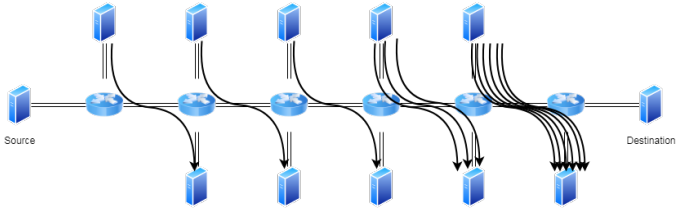

Figure 5: Single flow with a long path of variable queue occupancy.

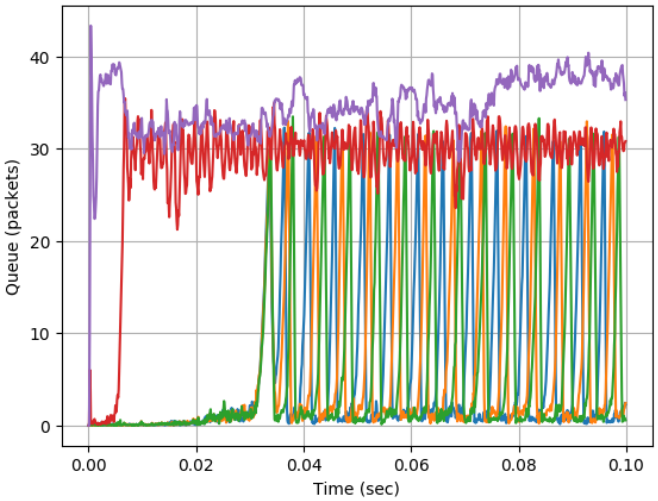

Figure 6: Queue occupancy of the buffers throughout the skinny topology experiments. The purple and red curves represent the multi-cross-flow links hence have relatively stable magnitude where as other curves, representing the non-bottleneck links, oscillate due to the bursty reactions of high rate cross-flows.

those links are smaller, so that they send relatively less bursty traffic into the network. The queue occupancy trends of the buffers in the network are shown in Figure 6.

RTT results and the total throughput for our benchmark flow are given in Figure 7. As discussed in section 3.2, RTT noise is omitted here. We observe that TIMELY flows attain

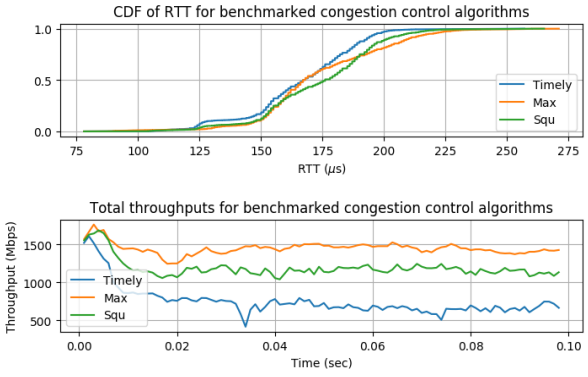

Figure 7: Primary flow RTT and throughput results for TIMELY (`SUM`), `MAX`, and `SUM_SQU` in the skinny topology simulation with oscillating non-bottleneck queues along the hops (figure 5). We assume no RTT measurement noise in this simulation (giving TIMELY an additional benefit).

allocations below their fair rate with the oscillating non-bottleneck queuing delays, which hurts overall throughput. The fair rate of the main flow is dictated by the link that has 6 other cross-flows (the bottleneck). However, because TIMELY flows interpret the oscillation in the non-bottleneck queues (single cross-flow links) as an increase in the delay, they drop their throughput even below the fair rate as shown in figure 7. TIMELY outperforms other schemes on RTT, but appears to be trading fairness and throughput for this improvement in latency.

3.2 Dissecting the RTT Signal

We now seek to understand the potential issues associated with behavior presented in section 3.1 and question the choice of RTT as a congestion control signal in the presence of oscillating non-bottleneck queues. To do this, we consider the round-trip hop-by-hop queue occupancy vector. This vector arguably represents the root congestion signal that other signals (ECN or RTT) can be derived from. For a path with k hops, we can write this vector as

$$(q_1, ..., q_{2k})$$

We can now consider multiple congestion signals as input to the gradient-based TIMELY algorithm. Some of possible functions are listed below.

- `SUM / TIMELY`: $f(q) = \sum_{i=1}^{2k} q_i$, the queuing delay which TIMELY attempts to estimate through precise RTT measures. This is the original TIMELY simulation, assuming no RTT measurement noise.
- `SUM_SQU`: $f(q) = \sqrt{\sum_{i=1}^{2k} q_i^2}$, a function of the queuing delay in which longer queues have an outsized impact without ignoring smaller queues
- `MAX`: $f(q) = \max_i q_i$, which only pays attention to the maximum queue size, i.e. bottleneck
- `IND`: $f(q) = \mathbb{I}\{\exists_i q_i > thresh\}$, the ECN signal formulation
- Functions which smoothly trade off a combination of these.

We focus our analysis on the TIMELY (`SUM`) formulation, the `SUM_SQU` function, which emphasizes congestion from bottleneck queues over non-bottleneck queues, and the `MAX` function, which entirely omits the signal from non-bottleneck queues.

As seen in figure 7, `MAX` and `SUM_SQU` seems to have larger RTT values compared to `SUM`. However, the throughput trend reveals that the lower RTT values of TIMELY (`SUM`) are achieved with lower throughput rates which are maintained throughout the simulations.

3.3 Simulations on Datacenter Topology

To perform a more realistic comparison of the characteristics of these functions, we now turn to a larger randomized datacenter topology. Our topology consists of 192 hosts that are connected to a 3-level CLOS network as shown in Figure 8. Every host randomly chooses another host to connect to and launches an FTP flow during a randomly selected time interval.

We benchmark RTT and throughput like in our previous experiments and present results in figure 9. The plots confirms our suspicion that TIMELY may trade throughput for smaller latency. The RTT plot shows the CDF for all of the RTT values that were observed by all flows in the simulation. TIMELY outperforms the other functions in RTT. However, it achieves lower total throughput than the `SUM_SQU` and `MAX` functions.

To evaluate the throughput-delay trade-off of these signals, we can consider measuring flow completion times, as suggested by [2], also depicted in figure 9. We consider three sizes of flows (50, 500, and 1500 packets for short, mid-length, and long flows respectively). Defining different size values would only change the times without significantly changing the ratio among algorithms. Our results suggest that the `MAX` function enables shorter flow completion times. We attribute this to the presence of non-bottleneck queues, but future work is required to validate this hypothesis in even larger topologies and outside of simulations.

4 DISCUSSION

We were able to reproduce TIMELY's claimed RTT measurements in a small scale simulation. Indeed, TIMELY achieved the lowest RTTs of all tested congestion control schemes across all of our experiments. However, the gains over DCTCP were not nearly as significant as claimed, since our DCTCP implementation (provided by the DCTCP authors) performed much better than the implemented DCTCP in Timely paper.

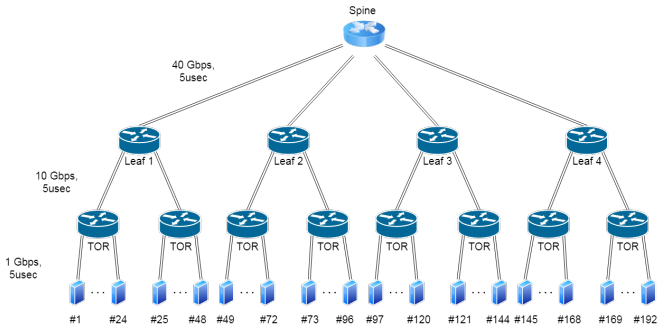

Figure 8: Large fat-tree topology for datacenter simulations.

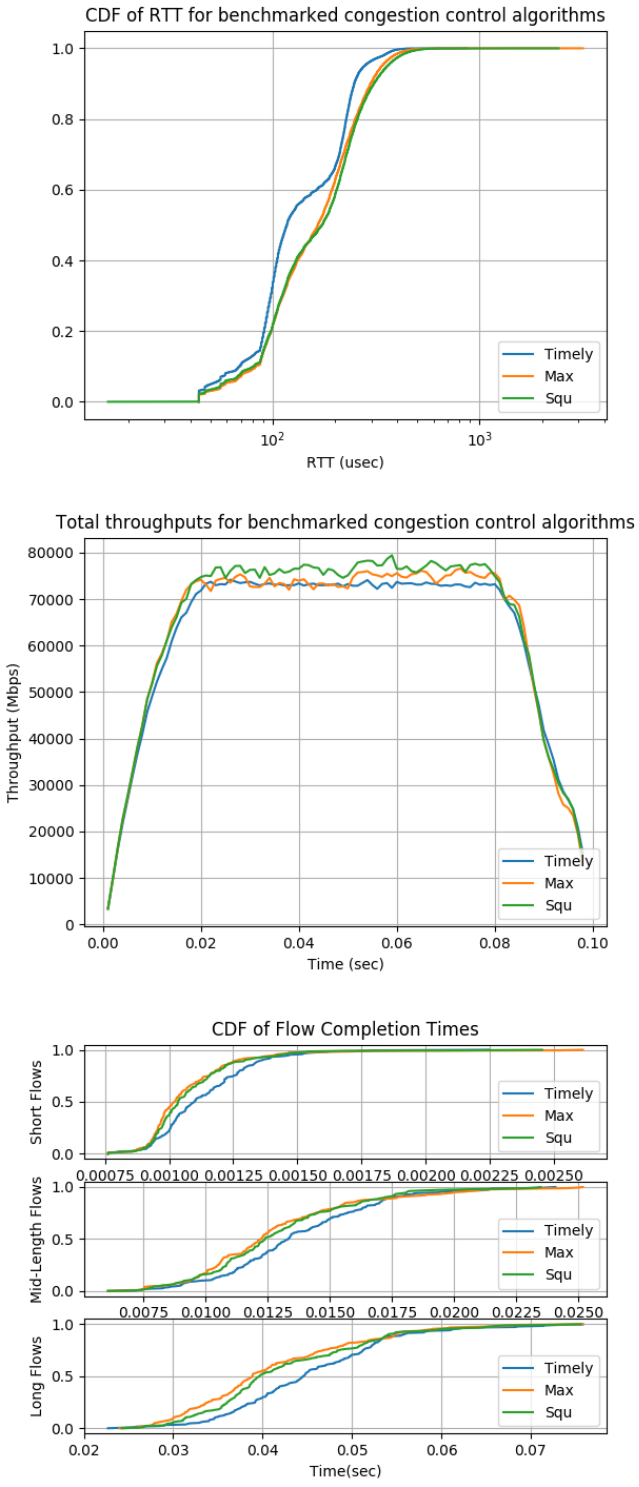

Figure 9: Results for datacenter simulation. The RTT plot shows the CDF for all of the RTT values that were observed by all flows in the simulation.

We discussed this discrepancy with TIMELY's authors, who pointed out that they had to create their own DCTCP implementation and pointed us to the limitations of this implementation, in particular the lack of OS bypassing. The upstream Linux DCTCP implementation arrived after their work was completed and was patched several times over the last two years. We conclude that TIMELY's RTT margin over DCTCP would likely be much lower in a datacenter implementation today.

Further, TIMELY's RTT gains appear to come at a cost that may have been underestimated by the original TIMELY paper. We observed that TIMELY allocated unfair flow rates both in the original topology and in our ablations. We saw that [12]'s "Patched TIMELY" algorithm mitigated this problem to some extent at the expense of more variable throughput overall, but this algorithm introduced an additional hyper-parameter that has to be tuned per topology, so conclusions about Patched TIMELY require further investigation. To this, TIMELY's authors replied that it is possible that some flows get a higher/lower share relative to the fair share rate due to the gradient methodology used. They did away with the gradient approach in later implementations of TIMELY.

When we attempted to measure TIMELY's performance in the presence of more variable queue occupancies along the path, we found that in a skinny topology and a larger datacenter-like topology, TIMELY appears to trade latency for throughput. We saw that TIMELY behaves rate-conservatively in the presence of noise in the RTT signal. We conjectured that this was because TIMELY's measurements contain both the signal from the bottleneck (good) and signals from all the lightly congested non-bottleneck queues (likely bad). Using hop-by-hop queue occupancy data, we tested whether the problem would be mitigated by paying less attention to non-bottleneck signals. We found that giving more importance to the larger queue occupancy values indeed allowed the client to achieve higher aggregate throughput and better flow completion times at a small cost in RTT.

Paying more attention to bottleneck signals makes intuitive sense. Any increase in other queues aren't due to any possible excessive transmission of the client in scope. Thus the sending rate or `cwnd` should not be decreased. Clients that pay less attention to the non-bottleneck signals don't need to back off with small rises in queue occupancy along the path. TIMELY (`SUM`) clients take the less-important information into consideration even if their fair share rates are not changing at all. As two alternative schemes, the `SUM_SQU` signal suppresses the effect of this information and `MAX` uses only the bottleneck queue occupancy information.

These schemes can be implemented in practice, for example using INT [6] on programmable switches. It is easy to imagine a datacenter setup in which switches stamp packets with the current queue occupancy directly when the packet

enters/leaves the switch or a scheme that only retains the max size. We do not claim that using TIMELY's gradient-based algorithm with the MAX signal is the final answer. Far from it, we believe that significant further work is required to design a tailored congestion control algorithm based on these functions of queue occupancy and consider fairness, stability, and more.

We also caution that the NS-2 experiments presented here are limited in nature. The cwnd discretization step in particular introduces several issues, increasing the gap between the set rate and observed throughput in several experiments. A future investigation of delay-based congestion control based on TIMELY's private hardware testbed would be of great value.

However, we believe that our results support questioning the effectiveness of using delay as the singular congestion signal. Based on our discussion with Google's and Microsoft's networking teams, we understand that TIMELY's focus on delay isn't for religious motivations but for practical considerations. Both parties agree that multi-bit ECN approaches could consolidate the agile feedback behavior of ECN and the precise measurements of delay. We HOPE to take the next step in that direction in a future project.

REFERENCES

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. 2010. Data Center TCP (DCTCP). *SIGCOMM Comput. Commun. Rev.* 40, 4 (Aug. 2010), 63–74. https://doi.org/10.1145/1851275.1851192

[2] Nandita Dukkipati and Nick McKeown. 2006. Why Flow-completion Time is the Right Metric for Congestion Control. *SIGCOMM Comput. Commun. Rev.* 36, 1 (Jan. 2006), 59–62. https://doi.org/10.1145/1111322.1111336

[3] Matthew P. Grosvenor. 2015. DCTCP NS-2 patch. https://github.com/camsas/qjump-ns2/. (2015). [Online; accessed May 2019].

[4] Stephen Ibanez and Kostis Kaffes. 2017. CS244 âĂŹ17: DCTCP âĂŞ DATA CENTER TCP. https://reproducingnetworkresearch.wordpress.com/2017/06/05/cs244-17-dctcp-data-center-tcp/. (2017). [Online; accessed May 2019].

[5] Teerawat Issariyakul and Ekram Hossain. 2011. *Introduction to Network Simulator NS2* (2nd ed.). Springer Publishing Company, Incorporated, Boston, MA.

[6] Changhoon Kim, Parag Bhide, Ed Doe, Hugh Holbrook, Anoop Ghanwani, Dan Daly, Mukesh Hira, and Bruce Davie. 2016. Inband Network Telemetry (INT). (2016). https://p4.org/assets/INT-current-spec.pdf

[7] Radhika Mittal. 2015. TIMELY Code Snippet. http://radhikam.web.illinois.edu/timely-code-snippet.cc. (2015). [Online; accessed May 2019].

[8] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. 2015. TIMELY: RTT-based Congestion Control for the Datacenter. *SIGCOMM Comput. Commun. Rev.* 45, 4 (Aug. 2015), 537–550. https://doi.org/10.1145/2829988.2787510

[9] Gaoxiong Zeng, Wei Bai, Ge Chen, Kai Chen, Dongsu Han, and Yibo Zhu. 2017. Combining ECN and RTT for Datacenter Transport. In *Proceedings of the First Asia-Pacific Workshop on Networking*. ACM, 36–42.

[10] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion control for large-scale RDMA deployments. *ACM SIGCOMM Computer Communication Review* 45, 4 (2015), 523–536.

[11] Yibo Zhu, Haggai Eran, Daniel Firestone, Chuanxiong Guo, Marina Lipshteyn, Yehonatan Liron, Jitendra Padhye, Shachar Raindel, Mohamad Haj Yahia, and Ming Zhang. 2015. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*. ACM, New York, NY, USA, 523–536. https://doi.org/10.1145/2785956.2787484

[12] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. 2016. ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY. In *Proceedings of the 12th International on Conference on Emerging Networking EXperiments and Technologies (CoNEXT '16)*. ACM, New York, NY, USA, 313–327. https://doi.org/10.1145/2999572.2999593