# ECN or Delay: Lessons Learnt from Analysis of DCQCN and TIMELY

Yibo Zhu[1], Monia Ghobadi[1], Vishal Misra[2], Jitendra Padhye[1]
[1]Microsoft [2]Columbia University

## ABSTRACT

Data center networks, and especially drop-free RoCEv2 networks require efficient congestion control protocols. DCQCN (ECN-based) and TIMELY (delay-based) are two recent proposals for this purpose. In this paper, we analyze DCQCN and TIMELY using fluid models and simulations, for stability, convergence, fairness and flow completion time. We uncover several surprising behaviors of these protocols. For example, we show that DCQCN exhibits non-monotonic stability behavior, and that TIMELY can converge to stable regime with arbitrary unfairness. We propose simple fixes and tuning for ensuring that both protocols converge to and are stable at the fair share point. Finally, using lessons learnt from the analysis, we address the broader question: are there fundamental reasons to prefer either ECN or delay for end-to-end congestion control in data center networks? We argue that ECN is a better congestion signal, due to the way modern switches mark packets, and due to a fundamental limitation of end-to-end delay-based protocols, that we derive.

## Keywords

Data center transport; RDMA; ECN; delay-based; congestion control

## 1. INTRODUCTION

Large cloud service providers are turning to Remote DMA (RDMA) technology to support their most demanding applications [21, 28, 31]. RDMA offers significantly higher bandwidth and lower latency than the traditional TCP/IP stack, while minimizing CPU overhead [7, 21, 31].

Today, RDMA is deployed using the RoCEv2 standard [18]. To ensure efficient operation, RoCEv2 uses Priority Flow Control (PFC) [17] to prevent packet drops due to buffer overflow. Since there are no packet drops, any end-to-end congestion control protocol for RoCEv2 networks must use either ECN markings, or delay as the congestion signal.

Last year, two protocols were proposed for this purpose, namely DCQCN [31] and TIMELY [21]. Though they are designed for enabling RoCEv2 in large data centers, their key assumptions are not RDMA-specific: data centers based on commodity Ethernet hardware, and no packet drops due to congestion. The difference is that DCQCN uses ECN marking as a congestion signal, while TIMELY measures changes to end-to-end delay.

In this paper, we analyze DCQCN and TIMELY for stability, convergence, fairness and flow completion time.

We have two motivations for this work. First, we want to understand the performance of DCQCN and TIMELY in detail. Given their potential for widespread deployment, we want to understand the tradeoffs made by the two protocols in detail, as well as offer guidance for parameter tuning. Second, using insights drawn from the analysis, we want to answer a broader question: are there fundamental reasons to prefer either ECN or delay as the congestion signal in data center networks?

To this end, we analyze DCQCN and TIMELY using fluid models and NS3 [32] simulations. Fluid models are useful for analyzing properties such as stability, and for rapid exploration of parameter space. However, fluid models cannot tractably model all features of complex protocols like DCQCN and TIMELY. Nor can fluid models compute measures like flow completion times. Thus, we also study the protocols using detailed, packet-level simulations using NS3. Our simulations in NS3 implement all known features of the protocols. We have released our NS3 source code at [1].

Before we proceed, we want to stress two things. First, we limit our discussion to ECN and delay as congestion signals and end-host rate-based control, since they are supported by commodity Ethernet switch and NIC hardware [21, 31]. Protocols that require hardware to do more [5, 8, 19] or use a central controller [24, 30] are outside the scope of this paper. We also do not consider designing a new protocol that would use both ECN and delay as congestion signals.

Second, it is not our goal to do a direct comparison of performance of DCQCN and TIMELY. Such comparison makes little sense, since both protocols offer several tuning knobs, and given a specific scenario, either protocol can be made to perform as well as the other. Instead, we focus on the core

*CoNEXT '16, December 12-15, 2016, Irvine, CA, USA*
© 2016 ACM. ISBN 978-1-4503-4292-6/16/12. . . $15.00
DOI: http://dx.doi.org/10.1145/2999572.2999593

behavior the two protocols to obtain broader insights. Our key contributions, and findings are summarized as follows.

**DCQCN:** $(i)$ We extend the fluid model proposed in [31], to show that DCQCN has a unique fixed point, where flows converge to their fair share. Using a discrete model, we also derive that the convergence speed is exponential. $(ii)$ We show that DCQCN is stable around this fixed point, as long as the feedback latency is low. The relationship between stability and the number of competing flows is, strangely, non-monotonic, which is very different from TCP's behavior [15]. DCQCN is stable for both very small, and very large number of flows, and tends to be unstable in between; especially if the feedback latency is high.

**TIMELY:** $(i)$ We develop a fluid model for TIMELY, and validate it using simulations. The model reveals that TIMELY can have infinite fixed points, resulting in arbitrary unfairness. $(ii)$ We propose a simple remedy, called Patched TIMELY, and show that the modified version is stable and exponentially converges to a unique fixed point.

**ECN or Delay:** $(i)$ We compare the flow completion time of TIMELY and DCQCN running the same traffic traces. DCQCN outperforms TIMELY due to better fairness, stability and fundamentally it is able to control the queue length better than TIMELY. Patched TIMELY closes the gap, but still cannot match DCQCN performance. We sweep the values of all DCQCN and TIMELY parameters and present the best combinations. Therefore, the performance difference is less about parameter tuning, but more likely due to the fundamental signal they use.

$(ii)$ Based on lessons learnt from analysis of DCQCN and TIMELY, we explain why ECN is a better signal for conveying congestion information. One reason for this is that modern shared buffer switches mark packets with ECN *at egress*, effectively decoupling the queuing delay from feedback delay. This improves system stability. Second, we present a *fundamental* result: for a distributed protocol that uses only delay as the feedback signal, you can achieve either fairness or a guaranteed steady-state delay, but not both simultaneously. For an ECN-based protocol you can achieve both by using a PI [14]-like marking scheme. This is not possible with delay-based congestion control. Finally, ECN signal is more resilient to jitter on the backward path as it only introduces delay in the feedback, whereas for delay based protocols jitter introduces delay *and* noise in the feedback signal.

## 2. BACKGROUND

The Remote Direct Memory Access (RDMA) technology offers high throughput (40Gbps or more), low latency (few $\mu$s), and low CPU overhead (1-2%), by bypassing the end-host kernels during data transfer. Instead, network interface cards (NICs) transfer data in and out of pre-registered memory buffers at the two end hosts.

Modern data center networks deploy RDMA using the RDMA over Converged Ethernet V2 (RoCEv2) [18] standard. RoCEv2 requires a lossless (or, more accurately, drop-free) L2 layer. Ethernet can be made drop-free using Priority Flow Control (PFC) [17]. PFC prevents buffer overflow on Ethernet switches and NICs. The switches and NICs track ingress queues. When the queue exceeds a certain threshold, a PAUSE message is sent to the upstream entity. The uplink entity then stops sending on that link till it gets an RESUME message. PFC is a blunt mechanism, since it does not operate on a per-flow basis. This leads to several well-known problems such as head-of-the-line blocking [31, 28].

PFC's problems can be mitigated using per-flow congestion control. Since PFC eliminates packet drops due to buffer overflow, either ECN or increase in RTT are the only two available "end-to-end" congestion signals. DCQCN relies on ECN, while TIMELY relies on RTT changes.

DCQCN and TIMELY are not RDMA-specific, so our analysis makes no reference to RoCEv2 or RDMA. DCQCN and TIMELY are not designed for wide area traffic, so our analysis focuses on intra-DC networks.

## 3. DCQCN

DCQCN is an end-to-end, rate-based congestion control protocol that relies on ECN [25]. It combines elements of DCTCP [2] and QCN [16]. DCQCN algorithm specifies behavior of three entities: the sender (called the reaction point (RP)), the switch, (called the congestion point (CP)), and the receiver (called the notification point (NP)). We now briefly describe the protocol; see [31] for more details.

**CP behavior:** At every egress queue, the arriving packet is ECN-marked [25] if the queue exceeds a threshold, using a RED [10]-like algorithm.

**NP behavior:** The NP receives ECN-marked packets and notifies the RP about it using Congestion Notification Packets (CNP) [18] Specifically, if a marked packet arrives for a flow, and no CNP has been sent for this flow in last $\tau$ microseconds, a CNP is generated immediately.

**RP behavior:** The RP adjusts its sending rate based on whether it receives a CNP within a period of time.

Upon getting a CNP, the RP reduces its current rate ($R_C$) and updates the value of the rate reduction factor, $\alpha$, like DCTCP, and remembers current rate as target rate ($R_T$) for later recovery, as follows:

$$\begin{aligned} R_T &= \quad R_C, \\ R_C &= \quad R_C(1-\alpha/2), \\ \alpha &= \quad (1-g)\alpha + g, \end{aligned} \tag{1}$$

If RP gets no feedback for $\tau'$, $\alpha$ is updated as:

$$\alpha = \quad (1-g)\alpha, \tag{2}$$

Note that $\tau'$ must be larger than the CNP generation timer.

RP increases its sending rate using a timer and a byte counter, in a manner identical to QCN [16]. The rate increases has two phases, five stages of so-called "fast recovery", where $R_c$ rapidly approaches $R_t$, and then a gradual additive increase. DCQCN does not have slow start. Senders start at line rate, in order to optimize the common case of no congestion. DCQCN relies on hardware rate limiters for per-packet rate limiting.

$$p(t) = \begin{cases} 0, & q(t) \le K_{\min} \\ \frac{q(t)-K_{\min}}{K_{\max}-K_{\min}} p_{\max}, & K_{\min} < q(t) \le K_{\max} \\ 1, & q(t) > K_{\max} \end{cases} \tag{3}$$

$$\frac{dq}{dt} = \sum_{i=1}^{N} R_C^{(i)}(t) - C \tag{4}$$

$$\frac{d\alpha^{(i)}}{dt} = \frac{g}{\tau'}\left(\left(1-(1-p(t-\tau*))^{\tau' R_C(t-\tau*)}\right) - \alpha^{(i)}(t)\right) \tag{5}$$

$$\begin{aligned} \frac{dR_T^{(i)}}{dt} = & -\frac{R_T^{(i)}(t) - R_C^{(i)}(t)}{\tau}\left(1-(1-p(t-\tau*))^{\tau R_C^{(i)}(t-\tau*)}\right) \\ & + R_{AI} R_C^{(i)}(t-\tau*) \frac{(1-p(t-\tau*))^{FB} p(t-\tau*)}{(1-p(t-\tau*))^{-B} - 1} \\ & + R_{AI} R_C^{(i)}(t-\tau*) \frac{(1-p(t-\tau*))^{FTR_C^{(i)}(t-\tau*)} p(t-\tau*)}{(1-p(t-\tau*))^{-TR_C^{(i)}(t-\tau*)} - 1} \end{aligned} \tag{6}$$

$$\begin{aligned} \frac{dR_C^{(i)}}{dt} = & -\frac{R_C^{(i)}(t)\alpha^{(i)}(t)}{2\tau}\left(1-(1-p(t-\tau*))^{\tau R_C^{(i)}(t-\tau*)}\right) \\ & + \frac{R_T^{(i)}(t) - R_C^{(i)}(t)}{2} \frac{R_C^{(i)}(t-\tau*) p(t-\tau*)}{(1-p(t-\tau*))^{-B} - 1} \\ & + \frac{R_T^{(i)}(t) - R_C^{(i)}(t)}{2} \frac{R_C^{(i)}(t-\tau*) p(t-\tau*)}{(1-p(t-\tau*))^{-TR_C^{(i)}(t-\tau*)} - 1} \end{aligned} \tag{7}$$

**Figure 1: DCQCN fluid model**

## 3.1 Model

A fluid model of DCQCN was described in [31]. The original model considers N flows with exactly same rates and states, traversing a single bottleneck link. In this paper, we extend it slightly to account for flows with different rates and states. Instead of having variables that are shared by all flows, we model each flow individually. The model is shown in Figure 1 and Table 1.

We assume that ECN marking is triggered before PFC [31] and does not delay due to PFC at current congestion point (see Section 5, ECN is marked on egress). Hence, we ignore the impact of PFC. Equation 3 calculates the probability of a packet getting marked. Equation 4 describes the queue behavior. Equation 5 captures the evolution of alpha. Equations 6 and 7 describe the calculation of target and sending rate, respectively.

**Model Validation:** In [31] it was shown that the fluid model matches actual hardware implementation. Here we only show that our NS3 packet-level simulations are in agreement with the model. To do so, we simulate and model a simple topology, in which N senders, connected to a switch, send to a single receiver, also connected to that switch. DCQCN parameters are set to the values proposed in [31]. Note that as per DCQCN specification, all flows start at line rate. Figure 2 shows that the fluid model and the simulator are in good agreement.

**Variables**

| | |
|---|---|
| $R_c$ | Current Rate |
| $R_t$ | Target Rate |
| $\alpha$ | See Equation (1) |
| $q$ | Queue Size |
| $t$ | Time |

**Parameters**

| | |
|---|---|
| $K_{min}, K_{max}, P_{max}$ | RED marking parameters. |
| $g$ | See Equation (1) |
| $N$ | Number of flows at bottleneck |
| $C$ | Bandwidth of bottleneck link |
| $F$ | Fast recovery steps (fixed at 5) |
| $B$ | Byte counter for rate increase |
| $T$ | Timer for rate increase |
| $R_{AI}$ | Rate increase step (fixed at 40Mbps) |
| $\tau$ | CNP generation timer |
| $\tau*$ | Control loop delay |
| $\tau'$ | Interval of Equation (2) |

**Table 1: DCQCN Fluid model variables and parameters**

## 3.2 Stability

We first obtain the fixed point of the system, and linearize the model around the fixed point. We analyze the linearized model for stability using standard frequency domain techniques [13].

THEOREM 1 (DCQCN'S UNIQUE FIXED POINT). *DCQCN has a unique fixed point of queue length and flow rates.*

PROOF. By setting the left-hand side of Equation 4 to 0, we see that any fixed points of the DCQCN (if they exist) must satisfy:

$$\sum_{i=1}^{N} R_C^{(i)}(t) = C \tag{8}$$

At any of the fixed points, we assume the value of $p$ is $p^*$, which is shared by all flows. The queue length and per-flow $\alpha^{(i)}$ at the fixed points are determined by Equation 3 and 5:

$$q^* = \frac{p^*}{p_{max}}(K_{max} - K_{min}) + K_{min} \tag{9}$$

$$\alpha^{(i)*} = 1 - (1-p^*)^{\tau' R_C^{(i)*}} \tag{10}$$

Next, we show that $p^*$ exists and is uniquely determined by $R_C^{(i)*}$ in the DCQCN model. Combining Equation 6 and 7, we eliminate the variable $R_T^{(i)*}$. After simplification, we see that the value of $p^*$ is determined by:

$$\frac{a^2\alpha^{(i)*}}{(b+d)(c+e)} = \tau^2 R_{AI} R_C^{(i)*} \tag{11}$$

Where we denote $a, b, c, d, e$ as follows:

$$a = 1-(1-p^*)^{\tau R_C^{(i)*}}, b = \frac{p^*}{(1-p^*)^{-B}-1}, c = \frac{(1-p^*)^{FB}p^*}{(1-p^*)^{-B}-1},$$
$$d = \frac{p^*}{(1-p^*)^{-TR_C^{(i)*}}-1}, e = \frac{(1-p^*)^{FTR_C^{(i)*}}p^*}{(1-p^*)^{-TR_C^{(i)*}}-1} \tag{12}$$

The LHS of Equation (11) is a monotonic function of $p$ when $p \in [0,1]$. Furthermore, when $p = 0$, the LHS is smaller

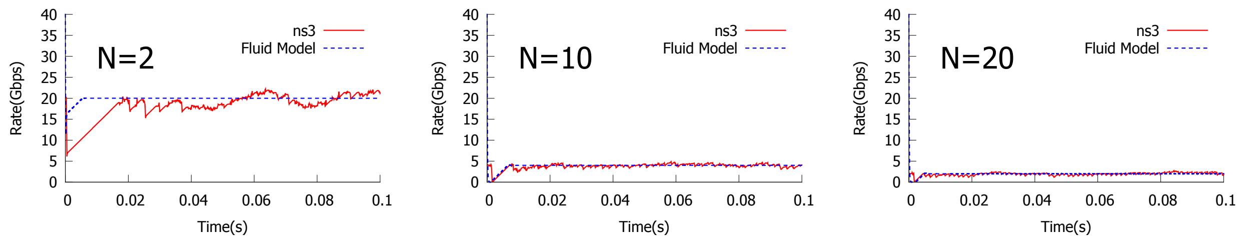
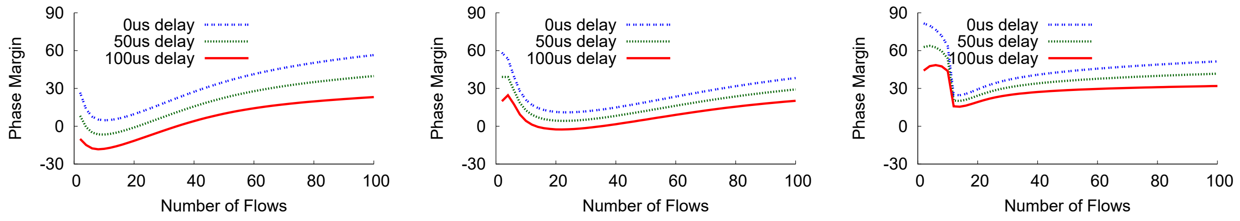


**Figure 2: Comparison of DCQCN fluid model and NS3 simulations**



(a) Default parameters ($R_{AI}$ = $40Mbps$, $K_{max} = 200KB$).

(b) $R_{AI} = 10Mbps$.

(c) $R_{AI}$ = $10Mbps$, $K_{max}$ = $1000KB$.

**Figure 3: DCQCN stability**

than RHS, and vice versa when $p = 1$. Thus DCQCN has a unique fixed point of marking probability $p^*$, leading to a unique fixed point of queue length $q^*$.

In § 3.3, we prove that flow rates do not reach a steady state until they converge to the same rate, *i.e.,*

$$R_C^{(1)*} = R_C^{(2)*} = ... = R_C^{(N)*} \tag{13}$$

The fixed point, where by definition the flow rates are steady, must satisfy this equation. Therefore, at the fixed point, $R_C^{(i)*} = \frac{C}{N}$, $i = 1, 2, ..., N$. $\square$

Next we approximate the value of $p^*$. Numerical analysis shows that $p^*$ is typically very close to 0. Therefore, we approximate the LHS of Equation 11 using Taylor series around $p = 0$. After omitting the $O(p^4)$ term, we have:

$$p^* \approx \sqrt[3]{\frac{R_{AI}N^2}{\tau' C^2}\left(\frac{1}{B} + \frac{N}{TC}\right)^2} \tag{14}$$

Therefore, the queue length $q^*$ is determined by $p^*$ (Equation 9), which depends on the number of flows $N$. A potential improvement is to make $q^*$ independent of $N$. With ECN as the signal, we may achieve this by using a PI-like control mechanism. See more discussion in Section 5.

**Stability analysis.** We test the system against *Bode Stability Criteria* [13]. The degree of stability is shown as *Phase Margin*. A stable system must have negative *Gain* (in dB) when there is a small oscillation aruond the fixed point, so that it converges back to the fixed point. *Phase Margin* is defined as how far the system is from the 0dB *Gain* state. The system is stable when its *Phase Margin* is larger than 0, and the larger *Phase Margin* means the system is more stable. *Phase Margin* is computed from the characteristic equation of a system. See Appendix A for details of the derivation of the characteristic equation for $R_C$. The numerical results are shown in Figure 3.

We analyze DCQCN stability in different conditions, particularly with different control signal delays $\tau*$, and different number of flows. An ideal protocol should be tolerant with control signal delays and scalable to any number of flows. In practice, $\tau*$ is dominated by propagation delay since ECN marking is not affected by the queuing delay at the congestion point (see Section 5). In data center environment, propagation delay is usually well below 100$\mu$s. There could be tens of flows competing for a single bottleneck. As Figure 3 shows, DCQCN, with default parameters, is mostly stable in such environment.

However, unlike TCP [20], the relationship between number of flows and the phase margin is non-monotonic. When the delay is large, *e.g., 100$\mu s$*, the phase margin dips below zero for certain number of flows, before rising again. For the set of parameters we have chosen, the system can be unstable with 10 flows at high feedback delays. DCQCN is increasingly stable with larger number of flows, which means good scalability. This point is further illustrated in the fluid model results shown Figure 4. When the feedback delay is small (4 $\mu$s), DCQCN is stable - flow rates, and queue length quickly[1] stabilizes regardless of the number of flows. However, when the delay is large (85$\mu$s), the protocol is unstable for 10 flows. It is, however, stable for 2 and 64 flows. Figure 5 shows the instability with packet-level simulations.

While this problem may not be particularly serious in practice, it can be easily fixed by tuning the values of $R_{AI}$ and $K_{max}$. Smaller $R_{AI}$ means flows increase their rate more gently, and stabilizes the system. Similarly, larger $K_{max} - K_{min}$ makes rate decreasing more fine grained, because the perturbation of queue length leads to smaller perturbation in marking probability. We show these trends in Figures 3(b) and 3(c). With small $R_{AI}$ and large $K_{max}$, DCQCN can be always stable even when the control signal delay reaches $100\mu s$, which equals to the propagation delay of a $30KM$

[1]Remember that DCQCN flows always start at line rate.

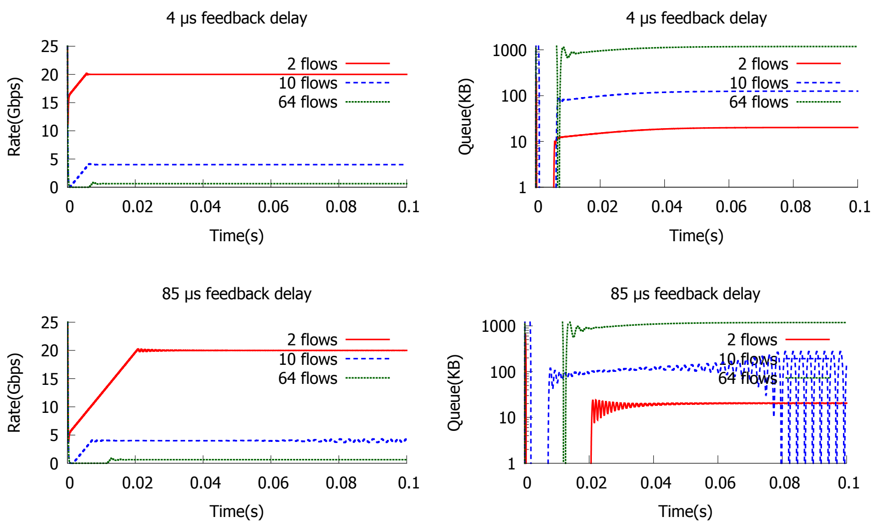


**Figure 4: Impact of delay and number of flows on DCQCN stability**

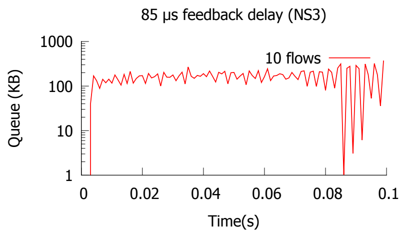


**Figure 5: NS simulations confirm lack of stability**

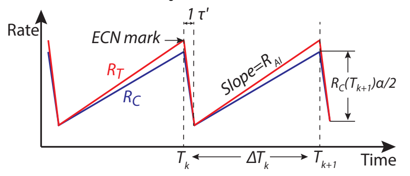


**Figure 6: DCQCN's AIMD-style updates of flow rate.**

cable, or $500KB$ queuing delay. Such large delays are rare in modern data center networks.

Note that tuning $R_{AI}$ and $K_{max}$ is a trade-off between stability and latency. Smaller $R_{AI}$ leads to slower ramp-up, while larger $K_{max}$ leads to larger queue length. In most cases, the default parameters strike a good enough balance between stability and latency.

## 3.3 Convergence

In Section 3.2, we showed that for reasonable parameter settings, DCQCN is stable *after* flows converge to the unique fixed point. We also showed that at the fixed point, the flows share the bandwidth equally. However, two questions remain unanswered: $(i)$ do flows the always converge to this fixed point, and $(ii)$ how fast do flows converge?

We cannot answer these questions using the fluid model, so like [3], we construct and analyze a discrete model of the rate adjustment at the RP. The default parameter settings given in [31] set both the Timer $T$ and $\alpha$ update interval $\tau'$ equal to $55\mu s$. Thus, we use $\tau'$ as the unit of time. The process of DCQCN rate update is similar to TCP AIMD, as shown in Figure 6[2]. The flows get the peak rates at $T_k$. For simplicity, we assume all flows are synchronized, and peak at the same time. This is a common assumption, for data center environments [3], especially for workloads like distributed storage system and MapReduce-like frameworks.

THEOREM 2 (DCQCN CONVERGENCE). *Under the control of DCQCN, the rate difference of any two flows decrease exponentially over time.*

PROOF. Here we provide a brief proof whereas more details can be found in Appendix B. Whenever a flow gets ECN marks at $T_k$, it reduces its rate in one unit of time, then starts $\Delta T_k - 1$ consecutive additive rate increases on $R_T^{(i)}$.[3] Here $\Delta T_k \triangleq T_{k+1} - T_k$. According to DCQCN's definition:

$$R_T^{(i)}(T_{k+1}) = \left(1 - \frac{\alpha^{(i)}(T_k)}{2}\right) R_C^{(i)}(T_k) + (\Delta T_k - 1)\, R_{AI} \tag{15}$$

$$\alpha^{(i)}(T_{k+1}) = (1-g)^{\Delta T_k - 1}\left((1-g)\alpha^{(i)}(T_k) + g\right) \tag{16}$$

For another flow, *e.g.*, the $j$th flow, we can simply rewrite Equation 16 by replacing $(i)$ with $(j)$. We subtract the equation of $j$th flow from the equation of $i$th flow:

$$\begin{aligned}&\alpha^{(i)}(T_{k+1}) - \alpha^{(j)}(T_{k+1}) = (1-g)^{\Delta T_k}\left(\alpha^{(i)}(T_k) - \alpha^{(j)}(T_k)\right)\\ &= \ldots = (1-g)^{\sum_{l=0}^{k}\Delta T_l}\left(\alpha^{(i)}(T_0) - \alpha^{(j)}(T_0)\right)\end{aligned} \tag{17}$$

This tells us the difference of $\alpha^{(i)}$ of *any* two flows will decrease exponentially. So $\alpha^{(i)}$ of different flows will converge to the same value. Once the $\alpha$ converged at some $T_{k'}$, we can show the rates $R_C$ converge afterwards. We rewrite the $j$th flow's Equation 15, and subtract it from Equation 15. Combining the analysis of $R_T$ in Appendix B, we get:

$$\begin{aligned}&R_C^{(i)}(T_{k+1}) - R_C^{(j)}(T_{k+1}) = \left(1 - \tfrac{\alpha(T_k)}{2}\right)\left(R_C^{(i)}(T_k) - R_C^{(j)}(T_k)\right)\\ &= \ldots = \prod_{l=k'}^{k}\left(1 - \tfrac{\alpha(T_l)}{2}\right)\left(R_C^{(i)}(T_{k'}) - R_C^{(j)}(T_{k'})\right)\end{aligned} \tag{18}$$

As long as $\alpha(T_k)$ has a lower bound that is greater than 0, the rates $R_C$ of different flows converge exponentially. In Appendix B, we prove that:

$$\alpha(T_0) > \ldots > \alpha(T_k) > \alpha(T_{k+1}) > \ldots > \alpha^* > 0 \tag{19}$$

where $\alpha^*$ is the fixed point of the Equation 16. Equation 19 concludes our proof. □

From Equation 18, we see that $R_C$ converges at the rate of at least $(1 - \frac{\alpha^*}{2})^k$, where $k$ is the number of AIMD cycles.

[2]Fast recovery does not change the nature of AIMD, since it can be combined with corresponding rate decrease as a multiplicative decrease event. As a common practice, like [4] does, we omit it in the following analysis.

[3]For simplicity, we omit hyper-increase and set $R_T = R_C$ upon rate decrease. This is a slightly simplified version of DCQCN.

**Algorithm 1** TIMELY rate calculation

```
 1: $newRTTDiff \leftarrow newRTT - prevRTT$
 2: $prevRTT \leftarrow newRTT$
 3: $rttDiff \leftarrow (1-\alpha) \cdot rttDiff + \alpha \cdot newRTTDiff$
 4: $rttGradient = rttDiff / D_{minRTT}$
 5: if $newRTT < T_{low}$ then
 6:     $rate \leftarrow rate + \delta$
 7: else if $newRTT > T_{high}$ then
 8:     $rate \leftarrow rate \cdot (1 - \beta \cdot (1 - T_{high}/newRTT))$
 9: else if $rttGradiant \leq 0$ then
10:     $rate \leftarrow rate + \delta$
11: else
12:     $rate \leftarrow rate \cdot (1 - \beta \cdot rttGradient)$
```

## 3.4 Summary

We have shown that DCQCN has a unique fixed point, where all flows get a fair share of the bottleneck bandwidth. For typical parameter values flows converge to this fixed point exponentially. DCQCN is generally stable around this fixed point, and $R_{AI}$ and $K_{max}$ can be tuned if needed. However, a PI-controller based approach may be the more principled way to ensure stability and fixed queue length.

## 4. TIMELY

TIMELY [21] is an end-to-end, rate-based congestion control algorithm that uses changes in RTT as a congestion signal. It relies on NIC hardware to obtain fine-grained RTT measurements. RTT is estimated once per completion event [18], which signals the successful transmission of a chunk (16-64KB) of packets. Upon receiving a new RTT sample, TIMELY computes new rate for the flow, as shown in Algorithm 1. If the new RTT sample is less than ($T_{low}$), TIMELY increases sending rate additively by $\delta$. If the new sample is more than ($T_{high}$), rate is decreased multiplicatively by $\beta$. If the new sample is between $T_{low}$ and $T_{high}$, the rate change depends on the RTT gradient. The gradient is defined as the normalized change between two successive RTT samples. If the gradient is positive (i.e. RTT is increasing), sending rate is reduced multiplicatively, in proportion to the RTT gradient. Otherwise, it is increased additively by $\delta$.

TIMELY flows do not start at line rate. If there are N active flows at a sender, a new flow starts at rate C/(N+1), where C is the interface link bandwidth [21].

## 4.1 Model

Our fluid model of TIMELY is shown in Table 2 and Figure 7. As before, $(i)$ we model N flows, traversing a single bottleneck link, and $(ii)$ and ignore the impact of PFC.

Equation 20 describes the queue behavior. Equation 21 describes rate computation. For simplicity, we ignore the hyperactive increase phase. When RTT is between$T_{low}$ and $T_{high}$, the rate computation depends on RTT gradient, which evolves according to Equation 22. The equation captures the EWMA filter, as well as normalization. Since the gradient is the difference between the current and the previous RTT sample, it depends on two queue lengths in past: one at time $t - \tau'$, and one at time $t - \tau' - \tau$. The value of $\tau'$ and $\tau*$ depend on past transmission rates, but to simplify the model,

**Variables**

| | |
|---|---|
| $R$ | Rate |
| $g$ | RTT gradient |
| $q$ | Queue Size |
| $t$ | Time |
| $\tau*$ | Rate update interval |
| $\tau'$ | Feedback delay |

**Parameters**

| | |
|---|---|
| $N$ | Number of flows at bottleneck |
| $C$ | Bandwidth of bottleneck link |
| $\alpha$ | EWMA smoothing factor |
| $\delta$ | Additive increase step |
| $\beta$ | Multiplicative decrease factor |
| $T_{low}$ | Low threshold |
| $T_{high}$ | High threshold |
| $D_{minRTT}$ | Minimum RTT for normalization |
| $D_{prop}$ | Propagation delay |
| $Seg$ | Burst size |

**Table 2: TIMELY fluid model variables and parameters**

we approximate their calculation as shown in Equations 23 and 24. Equation 23 captures the fact that the TIMELY implementation gets RTT feedback once per burst, and rate updates are scaled by $D_{minRTT}$ to ensure that rate update frequency is limited (See Section 5 of [21]).

The fluid model (by its very nature) essentially assumes smooth and continuous transmission of data. The TIMELY implementation is more bursty, since rate is adjusted by modulating gaps between transmission of 16 or 64KB chunks; while the chunks themselves are sent at near-line rate [21]. TIMELY designers made this decision for engineering reasons - they wanted to avoid taking dependence on hardware rate limiters. We will return to this point later.

$$\frac{dq}{dt} = \sum_i R_i(t) - C \tag{20}$$

$$\frac{dR_i}{dt} = \begin{cases} \frac{\delta}{\tau_i^*}, & q(t-\tau') < C * T_{low} \\ \frac{\delta}{\tau_i^*}, & g_i \leq 0 \\ -\frac{g_i \beta}{\tau_i^*} R_i(t), & g_i > 0 \\ -\frac{\beta}{\tau_i^*}(1 - \frac{C*T_{high}}{q(t-\tau')}) R_i(t), & q(t-\tau') > C * T_{high} \end{cases} \tag{21}$$

$$\frac{dg_i}{dt} = \frac{\alpha}{\tau_i^*}(-g_i(t) + \frac{q(t-\tau') - q(t-\tau'-\tau_i^*)}{C * D_{\min RTT}}) \tag{22}$$

$$\tau_i^* = \max\{\frac{Seg}{R_i}, D_{\min RTT}\} \tag{23}$$

$$\tau' = \frac{q}{C} + \frac{MTU}{C} + D_{prop} \tag{24}$$

**Figure 7: TIMELY fluid model**

**Model Validation:** Since we do not have access to TIMELY implementation, Figure 8 compares the TIMELY fluid model with the NS3 packet-level simulations, using parameter values recommended[4] in [21]. The simulator can model both per-packet pacing, as well as the bursty behavior of TIMELY

[4] $C = 10Gbps$, $\beta = 0.8$, $\alpha = 0.875$, $T_{low} = 50\mu$s, $T_{high} = 500\mu$s, $D_{minRTT} = 20\mu$s.

implementation; here we use per-packet pacing. As before, we model N senders connected to a switch, sending to a single receiver connected to the same switch. The starting rate for each flow is set to be 1/N of the link bandwidth. We see the fluid model and the simulator are in good agreement.

## 4.2 Analysis

We now show that TIMELY, as described in Algorithm 1 has *no fixed point*. The implication is that the queue length never converges, nor do the sending rates of the flows. The system operates in limit cycles, always oscillating. Moreover, while the system is oscillating, there are *infinite* solutions for the sending rates of the flows that satisfy the fluid equations at any point. So, even if we can limit the magnitude of the limit cycles by choosing parameters carefully, we can make no claims on the fairness of the protocol since the system could be operating at any of those infinite solutions.

THEOREM 3 (NO FIXED POINT FOR TIMELY). *The system described in Figure (7), has no fixed points.*

PROOF. We prove the result by contradiction. At the fixed point, all differential equations converge to 0. Thus:

$$dq/dt = 0 \text{ and } \sum_i R_i(t) = C \tag{25}$$

Now, either $g_i > 0$ or $g_i \leq 0$. If $g_i \neq 0$, then:

$$\frac{dg_i}{dt} = \frac{\alpha}{\tau_i^*}(-g_i(t) + \frac{q(t-\tau') - q(t-\tau'-\tau_i^*)}{C * D_{\min RTT}}) = -\frac{\alpha}{\tau_i^*}g_i(t) \neq 0 \tag{26}$$

Thus, $g_i$ is zero for $dg_i/dt$ to be zero. But then:

$$dR_i/dt = \delta/\tau_i^* \neq 0 \tag{27}$$

Thus, all derivatives cannot be simultaneously 0 and thus the system has no fixed point. □

If we modify the fluid model very slightly, by moving the equality condition to the term involving $g_i$, we get:

$$\frac{dR_i}{dt} = \begin{cases} \frac{\delta}{\tau_i^*}, & q(t-\tau') < C * T_{low} \\ \frac{\delta}{\tau_i^*}, & g_i < 0 \\ -\frac{g_i \beta}{\tau_i^*}R(t), & g_i \geq 0 \\ -\frac{\beta}{\tau_i^*}(1 - \frac{C * T_{high}}{q(t-\tau')})R(t), & q(t-\tau') > C * T_{high} \end{cases} \tag{28}$$

This is equivalent to changing the $\leq$ sign on line 9 of Algorithm 1 to $<$. This makes little difference in practice, since floating point computations for $g_i$ rarely yield an exact zero value – we have verified this via simulations. With this modification, we can obtain the condition that with $g_i = 0$, $\frac{dq}{dt} = 0$, $\frac{dg_i}{dt} = 0$ and $C * T_{low} < q < C * T_{high}$. However, now we run into the issue that TIMELY moves from zero fixed points to *infinite* fixed points!

THEOREM 4 (INFINITE FIXED POINTS). *The system described by Figure (7), with modification introduced in Equation 28 has infinite fixed points*

PROOF. To obtain $dg_i/dt = 0$, we need $g_i = 0$ and $dq/dt = 0$. Note that $q$ cannot converge to a value outside of the thresholds $C * T_{low}$ and $C * T_{high}$ as that would imply $dR_i/dt \neq 0$.

Any value of $q$ such that $C * T_{low} < q < C * T_{high}$ makes $dR_i/dt = 0$ for any value of $R_i$ as long as $\sum_i R_i(t) = C$ and hence $q$ and $R_i$ have infinitely many fixed points. □

There is no requirement that at the fixed point $R_i = C/N$. In fact, $R_i/R_j, i \neq j$ is not even bounded, so we cannot make any claims on the fairness of TIMELY. Thus the fixed point of TIMELY is entirely unpredictable. This is borne out by the simulation results shown in Figure 9, where we only change the start time and initial rates of two flows, keeping everything else constant, and we end up in completely different operating regimes.

**Impact of per-burst pacing:** This analysis begs the question – why does TIMELY work well in practice, as reported in [21]? The answer appears to lie in the fact that the TIMELY implementation does not use hardware rate limiters. Instead, the TIMELY implementation controls rate by adjusting delay between successive transmissions of chunks that can be as large as 64KB. Each chunk is sent out at or near line rate.

The results shown in Figure 9 were obtained with per-packet pacing. If, instead we user per-burst pacing, TIMELY appears to converge, as shown in Figure 10(a). The bursts introduce enough "noise" to de-correlate the flows, and this appears to lead the system to a relatively stable fixed point. We attempted to mathematically prove that per-burst pacing would lead to a unique fixed point, but were unable to do so.

In any case, per-burst pacing is not ideal, since it can lead to large oscillations in queue length, leading to poor utilization. This is apparent in Figure 10(b), where we use 64KB chunks. The initial chunks sent by the two senders arrive at the switch near-simultaneously (i.e. "incast"), and both flows receive a very large RTT sample. This causes TIMELY to reduce its rate drastically (line 8 in the TIMELY algorithm). Since the subsequent rate increase occurs in small steps ($\delta = 10Mbps$, see line 6.)[5], it takes a long time for the flow rates to climb back to their fair share.

These problems can be mitigated to some extent by sending bursts at less than line rate[6], by adjusting the burst size, or by adjusting the $T_{min}$ threshold. However, such tuning is fragile, since the right values of these parameters depend not just on the link speed, but also on the number of competing flows, which is unknown at the time of configuration.

In summary, while burst pacing can lead to a fixed point by introducing noise, it can lead to other problems. The fact that TIMELY cannot maintain a stable queue length has a detrimental impact on the flow level performance (*e.g.,* flow completion times), especially at higher percentiles. We illustrate this in Section 5.1.

Rather than rely on "noise" to ensure convergence and stability, we propose a simple fix to the TIMELY algorithm.

[5]HAI kicks in only after $RTT > T_{low}$. See Algorithm 1 in [21]

[6]Indeed, the TIMELY does this, see § 5 in [21].

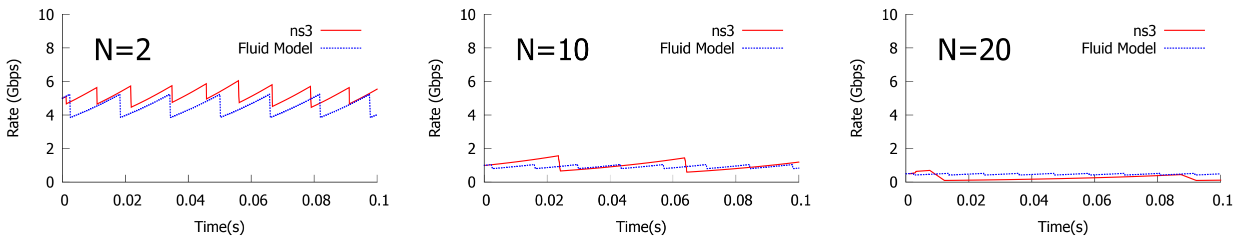
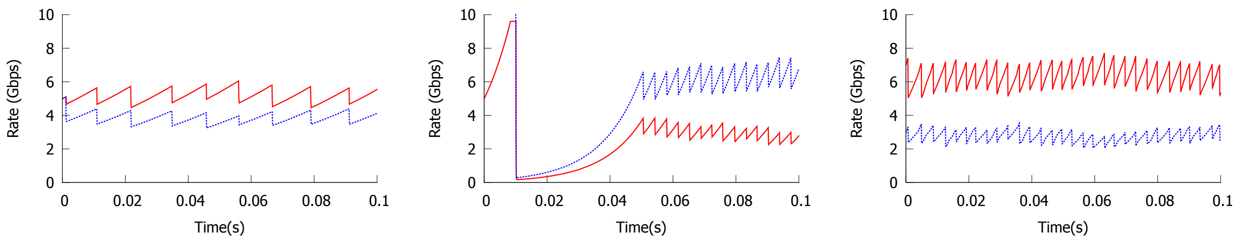


**Figure 8: Comparison of TIMELY fluid model and simulations**



(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

**Figure 9: Performance of two TIMELY flows under different starting conditions**

**Algorithm 2** Patched TIMELY rate calculation

```
1: $newRTTDiff \leftarrow newRTT - prevRTT$
2: $prevRTT \leftarrow newRTT$
3: $rttDiff \leftarrow (1-\alpha) \cdot rttDiff + \alpha \cdot newRTTDiff$
4: $rttGradient = rttDiff / D_{minRTT}$
5: if $newRTT < T_{low}$ then
6:     $rate \leftarrow rate + \delta$
7: else if $newRTT > T_{high}$ then
8:     $rate \leftarrow rate \cdot (1 - \beta \cdot (1 - T_{high}/newRTT))$
9: else
10:    $weight \leftarrow w(rttGradient)$
11:    $error \leftarrow \frac{newRTT - RTT_{ref}}{RTT_{ref}}$
12:    $rate \leftarrow \delta(1-weight) + rate \cdot (1 - \beta \cdot weight \cdot error)$
```

## 4.3 Patched TIMELY

In order to ensure there is a unique fixed point, and all flows get fair share and are stable at the fixed point, we make two minor modifications over TIMELY, as shown in Algorithm 2. We only modify the last four lines of Algorithm 1.

First, we make the step of rate decrease rely on absolute RTT, instead of the gradient of RTT. In effect, this means that all flows have the knowledge of the bottleneck queue length. This ensures two things. First, the system can have a unique fixed point, determined by the RTT[7]. Second, all flow can converge to the same rate, since they share the knowledge of the bottleneck RTT. The side effect, of course, is that, with different number of flows, the fixed point of queue can be different. We will address this in §5.

Second, we use a continuous weighting function $w(g)$ to make the transition between rate increase and rate decrease smooth. This avoids the on-off behavior that causes oscillation. This is similar to the fact that probabilistic ECN marking stabilizes TCP [20], DCTCP [3], QCN [4] and DCQCN. With $w(g)$, we combine the two conditions of $g \leq 0$ and $g > 0$ in the $dR(t)/dt$ equation:

$$\frac{dR_i}{dt} = \begin{cases} \frac{\delta}{\tau*}, & q(t-\tau') < C * T_{low} \\ \frac{(1-w_i)\delta}{\tau*} - \frac{w_i \beta R_i(t)}{\tau*}\frac{q(t-\tau')-q'}{q'}, & Otherwise \\ -\frac{\beta}{\tau*}(1 - \frac{C*T_{high}}{q(t-\tau')})R_i(t), & q(t-\tau') > C * T_{high} \end{cases} \quad (29)$$

where $w_i$, the weight of rate decreasing, is a function of $g_i$, and must satisfy $0 \leq w_i(g_i) \leq 1$ for any $g_i$. Intuitively, $w_i(g_i)$ is monotonically increasing with $g_i$, because larger RTT gradient should lead to larger rate decrease. In original TIMELY protocol, $w_i(g_i)$ is an indicator function of $g_i$, *i.e.*, $w_i(g_i) = 1$ when $g_i \geq 0$, and $w_i(g_i) = 0$ when $g_i < 0$. Here we simply use a linear function of $g_i$ for $w_i$:

$$w_i = \begin{cases} 0, & g_i \leq -\frac{1}{4} \\ 2g_i + \frac{1}{2}, & -\frac{1}{4} < g_i < \frac{1}{4} \\ 1, & g_i \geq \frac{1}{4} \end{cases} \quad (30)$$

In Equation 29, $q'$ is a reference queue length. We simply set it as $C * T_{low}$, so that we decrease the rate faster if the queue length exceeds $C * T_{low}$. All other TIMELY parameters remain the same except we set $\beta = 0.008$ and $Seg = 16KB$. We prove that this patched TIMELY protocol has desirable stability and convergence properties that original TIMELY does not guarantee:

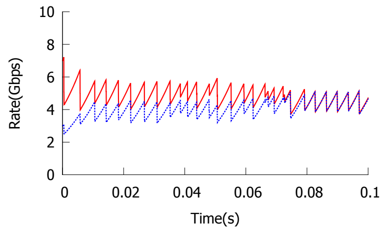
THEOREM 5 (PATCHED TIMELY'S FIXED POINT.). *The system described in Equation 29 has a unique fixed point. All flows have the same rate at this fixed point, and the queue length is:*

$$q^* = \frac{N\delta q'}{\beta C} + q' \quad (31)$$

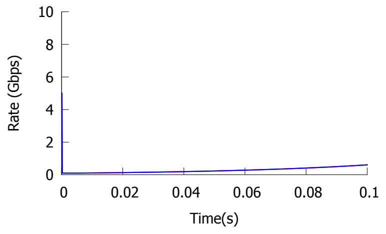
*The system described in Equation 29 always exponentially converges to the unique fixed point.*

The detailed proof is similar to the proof of Theorem 2 and omitted due to the lack of space.

[7]Recall that the issue with original TIMELY was that RTT gradient could be the same, but absolute RTTs could be different.



(a) Two flows starting at 7 and 3Gbps, 16KB burst



(b) Two flows starting at 5Gbps, 64KB burst

**Figure 10: TIMELY with bursts**

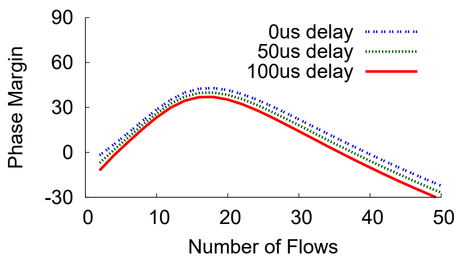
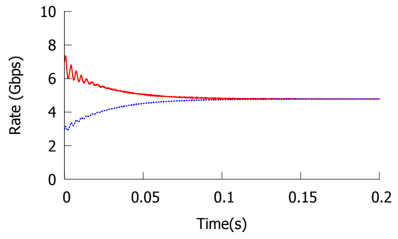
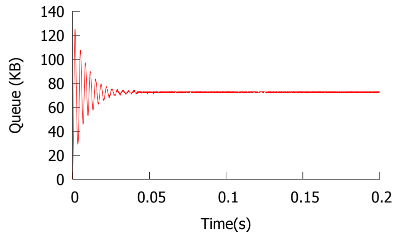


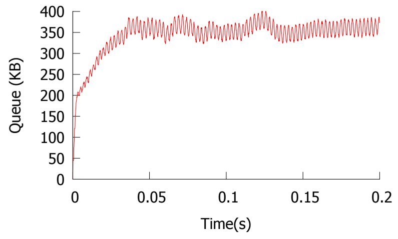
**Figure 11: Patched TIMELY stability**



(a) Two flows, 7Gbps and 3Gbps



(b) Two flows, 7Gbps and 3Gbps



(c) 40 flows, starting at 0.25Gbps

**Figure 12: Performance of patched TIMELY**

We further verify patched TIMELY convergence and stability using simulations. Figure 12(a) and 12(b), shows that flows with different initial rates converge to the fixed point and are stable without oscillation, opposed to Figure 9(c). Results for the case depicted in Figure 9(b) are similar.

**Stability.** We proceed as we did for DCQCN – linearize the equations, Laplace transform and compute the phase margin of its characteristic equation. The phase margin result shows this system is stable until the number of flows is greater than 40 (Figure 11). This is again confirmed by NS-3 simulation (Figure 12(b) and 12(c)). After 40 flows, the phase margin falls below 0 rapidly because more flows lead to larger queue size (see Equation 31), thus leading to larger feedback delay (see Equation 24). This leads to system instability. In general, with some minor tuning, TIMELY can be stable within a range of number of flows.

## 4.4 Summary

We showed that the TIMELY protocol, as proposed in [21] can have infinite fixed points. We proposed a small fix to address the problem, and showed that the resulting protocol converges to a fixed point, where all flows share the bottleneck bandwidth equally. The protocol is stable around this fixed point, as long as the number of flows is not too high.

## 5. ECN VERSUS DELAY

We have now seen that both DCQCN and TIMELY (with a small modification) can achieve desirable properties like fairness, stability and exponential convergence, if properly tuned for a given scenario. Can we thus conclude that ECN and delay are both "equivalent" signals, when it comes to congestion control?

We believe that the answer is no. As an example, we compare the performance of DCQCN (ECN-based) and TIMELY (delay-based), and find that DCQCN outperforms TIMELY. We then discuss the fundamental reasons why ECN-based protocols are better than delay-based protocols.

### 5.1 Case study: DCQCN versus TIMELY on flow completion time

While we show that both ECN-based DCQCN and delay-based TIMELY can be stable and fair if properly tuned (see Section 3 and Section 4), they may differ on other important performance metrics. For example, the end users often care about flow completion times, especially for short flows [8]. We compare the flow completion times of DCQCN and TIMELY with a simple simulation, using the classic dumbbell topology shown in Figure 13. The topology consists of 20 nodes – 10 senders and 10 receivers. All traffic flows across the bottleneck link between the two switches, SW1 and SW2. All links are 10Gbps with $1\mu$s latency.

The traffic consists of long and short-lived flows, between pairs of randomly selected sender and receiver nodes. The flow size distribution is derived from the traffic distribution reported in [2]. The interarrival time of flows is picked from am exponential distribution. The load on the bottleneck link is varied by changing the mean of the distribution. This traffic generation model was also used in several recent studies, including pFabric [5] and ProjecToR [12].

Both DCQCN and TIMELY used the default parameter settings recommended in [31] and [21], respectively. The metric of interest is the flow completion time of small flows. Following pFabric [5], we define small flows as flows that send fewer than 100KB. We also tune and test different protocol parameter and small flow threshold settings. The results are similar.

Figure 14 shows the median and 90th percentile of FCT and DCQCN, TIMELY original and patched TIMELY as the load is varied. Patched TIMELY is our modification to TIMELY's protocol to ensure a unique fixed point (§4.3).

The X axis shows relative load: load factor of 1 corresponds to an average of 8Gbps of traffic on the bottleneck link. The scaling is linear. We see that at higher loads, FCT for both TIMELY and patched TIMELY is high, and highly variable. This illustrated in detail in Figure 15, which shows the CDF of the flow completion time for load factor of 0.8.

The reason for TIMELY's poor performance is evident from Figure 16, which shows the queue length at the link between SW1 and SW2 for a load factor of 0.8. As shown, the queue length under TIMELY can grow to a very high value, and is highly variable. In contrast the DCQCN queue has a fixed point between the RED thresholds and even in the transient state the queue stays within the bounds. Note that patched TIMELY is operating in between the original TIMELY protocol and DCQCN. This is because our fix is ensuring a unique fixed point (and thus fairness), without changing the dynamics of TIMELY's queue build up.

We note that in all cases, the link utilization is roughly the same for DCQCN and TIMELY, indicating that the long flows performed similarly with both schemes.

## 5.2 ECN advantages

As shown above, despite that both protocols are fair and stable, TIMELY has larger queue dynamics than DCQCN, which leads to worse short flow completion time. We believe this is due to the following reasons.

**ECN marking is done on packet egress, thus has faster response:** Modern shared-buffer switches, especially those that use Broadcom's merchant silicon, do ECN marking on *packet egress*. When a packet is ready to depart, the controller checks the egress queue for that port *at that instant*, and marks the packet according to the specified algorithm (e.g. Equation 3). Thus, the mark always conveys information about the state of the queue at the time of packet *departure*, even if the egress queue is long.

RTT measurements are different: If the egress queue discipline is FIFO within a priority class (which it typically is), the delay experienced by a packet reflects the state of the queue at the time the packet *arrives* at the queue. This means that the control signal carried by the ECN mark is delayed only by the propagation delay, but the control signal carried by the RTT signal is delayed both by the queuing delay as well as the propagation delay.

This is a subtle difference: the claim is not that ECN carries more information; it that the delay of the control loop is decoupled from the queuing delay.

This is why the DCQCN fluid model (Figure 1) assumes that the control loop delay is constant. DCTCP fluid model makes the same assumption [3]. We cannot make the same assumption for TIMELY, and thus we incorporate $\tau'$ and Equation 24 in the TIMELY fluid model (Figure 7).

This means that as the queue length increases (e.g. when there are more flows), congestion control algorithms that rely on RTT suffer from increasing *lag* in their control loop, making them more difficult to control. We see this happening for TIMELY (Figure 11). DCQCN is affected less by this effect (Figure 3(a)). To further confirm that ECN marking on egress is important for stability, we run DCQCN with ECN marking on ingress for comparison. Figure 17 shows that marking on ingress leads to queue length fluctuation.

Researchers have observed similar "bufferbloat" [11] problems in the wide area networks solutions such as LEDBAT [26] have been proposed. Similarly, in modern data center networks, queuing delays can easily dominate switching and propagation delays. For example, an Arista 7050QX32 has 40Gbps ports, and a total shared buffer of switch has 12MB. Even if just 1MB worth of queue builds up at an egress port,[8] it takes 200 $\mu$s to drain it. In contrast, the one-hop propagation delay, with cut-through forwarding, is around 1-2 $\mu$s. Typical DC network diameter is 6 hops, so overall propagation delay is under $25\mu s$.

The reader may argue that it is easy to fix this issue – all we have to do is to build a delay-based protocol that strives to keep the bottleneck queue (more-or-less) constant. Then, the control signal delay experienced by the RTT feedback signal is also fixed, albeit a little higher (propagation delay, plus fixed queuing delay).

However, a delay based congestion control protocol that maintains a fixed queue, cannot ensure fairness.

**For delay-based control, fixed queue comes at the cost of fairness:** One way to build protocols that guarantee delay to a fixed quantity is to use a controller with *integral* control [14, 6]. The idea behind integral control is to look at an error signal, *e.g.,* the difference between the actual queue length and a desired or reference queue length, and adjust the feedback until the error signal goes to 0. A stable PI controller is guaranteed to achieve this. In a continuous system, the feedback signal $p(t)$ evolves in the following way with a PI controller:

$$\frac{dp}{dt} = K_1\frac{de}{dt} + K_2 e(t) \tag{32}$$

When the feedback signal converges, both the error signal $e(t)$ as well as the derivative of the error signal, $de/dt$ must converge to 0. The derivative of the error signal, (the derivative of the queue length), goes to 0 when the sum of the rates $R_i$ match the link capacity $C$. The error signal itself goes to 0 when the queue length matches the reference value. Thus the integral controller implements the "match rate, clear buffer" scheme presented in [6].

For DCQCN we can implement the PI controller to mark the packets at the switch instead of RED (which is a proportional controller without the integral action) and use that $p$ in the usual way to perform the multiplicative decrease.

For (patched) TIMELY, we can measure the delay at the end hosts and implement a PI controller by generating an internal variable "$p$", using the error signal "$e(t)$" as the difference between the measured delay and some desired delay. This internal variable $p$ can then replace the $\frac{q(t-\tau')-q'}{q'}$ term in Equation (29) as the feedback to control the rates.

We implemented the PI controller for both the DCQCN and patched TIMELY fluid models and performed simulations. As we see in Figure 18 for DCQCN, all the flows

[8]This requires enabling dynamic thresholding, but it is almost always enabled in real deployments.

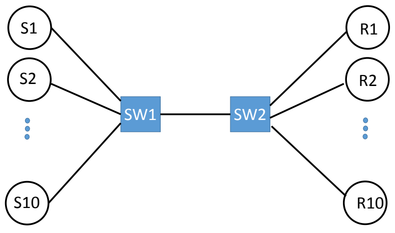


**Figure 13: The dumbbell topology. All links are 10Gbps with 1$\mu$s latency.**

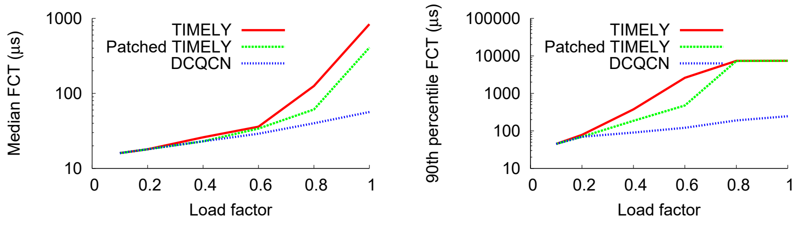


**Figure 14: Median and 90th percentile of FCT of small flows. Note log scale on Y axis.**

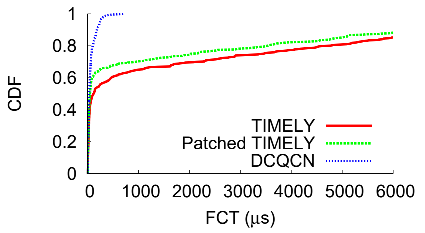


**Figure 15: CDF of FCT for load=0.8**

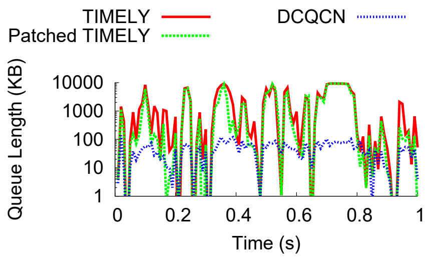


**Figure 16: Bottleneck Queue for load=0.8. Note log scale on Y axis.**

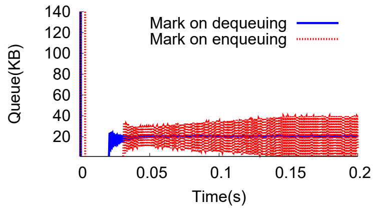


**Figure 17: DCQCN stability when two flows compete for a bottleneck with 85$\mu$s feedback delay.**

converge to the same (fair) rate and the queue length is stabilized to a preconfigured value, regardless of the number of flows (as well as regardless of propagation delay). This is important not only for stability, but also for performance reasons in a data center networks, where is important to ensure that completion times for short flows do not suffer from excessive queuing delays [2].

In contrast, when we use a PI controller at the end hosts with patched TIMELY, we see that although we can control the queue to a specified value (300 KB), we cannot achieve fairness (Figure 19). Thus, while patched TIMELY was able to achieve fairness without guaranteeing delay, with PI it is able to guarantee delay without achieving fairness.

We next prove a result that formalizes this fundamental tradeoff between fairness and guaranteed steady-state delay for protocols that rely on delay measurements at the end points to implement congestion control. We first assume that the steady state throughput achieved by a congestion control transport protocol is a function of the observed delay $d$ and some feedback value $p$. The value of $p$ can be the loss probability or the ECN marking probability or some internal variable $p$ computed by the patched TIMELY+PI mechanism we described above. Thus, $R = f(d,p)$ for steady state throughput $R$ and some function $f(d,p)$. Then the following theorem formalizes the fairness/delay tradeoff in such systems.

THEOREM 6 (FAIRNESS/DELAY TRADEOFF). *For congestion control mechanisms that have steady state throughput of the kind $R = f(d,p)$, for some function $f$, delay $d$ and feedback $p$, if the feedback is based on purely end to end delay measurements, you can either have fairness or a fixed delay, but not both simultaneously.*

PROOF. To guarantee fairness, the system must have a unique fixed point. Consider $N$ flows sharing a link of capacity $C$. Then, for each flow, we have $R_i = f(d,p_i), i = 1,\ldots,N$. There is an additional equation constraining the throughput at the link, $\sum R_i = C$. Hence we have $N+1$ equations and $2N$ unknowns – $\{R_i,p_i\}, i = 1\ldots,N$. This is an underdetermined system with infinite (or no) solutions. To make this system consistent, we need a common $p_i$, reducing the number of variables to $N+1$. That can be achieved either by marking at the switch (violating the assumption of delay being the only feedback), or by making this $p_i$ a function of the (common) queue length. However if we control the delay to a fixed quantity, it becomes agnostic of the number of flows which will make the system of equations inconsistent, since the constraint $\sum_i R_i = C$ implies the steady state throughput of a flow depends on the number of flows contending. Thus, to make the system consistent $p_i$ has to be a function of the common queue length which depends on the number of flows and hence we cannot control the delay to a fixed quantity. $\square$

**Remarks:** The results of this theorem are generic to congestion control protocols that use delay or ECN as feedback and are not RDMA or TIMELY/DCQCN specific. Note that a **RED** style AQM scheme can guarantee a unique fixed point that depends on the number of flows. However, given the low delay requirements in a data center environment, that would require the RED marking profile to have a steep slope (marking probability that goes from 0 to 1 over a small buffer space). As shown in [15], a steep slope leads to an unstable controller leading to oscillations in the flow rates, increased jitter and loss of utilization. Also, the preceding result does not apply to systems with limit cycles,[9] with which rate-based protocols do not have steady state throughput.

[9]Some window-based protocols have limit cycles [3].

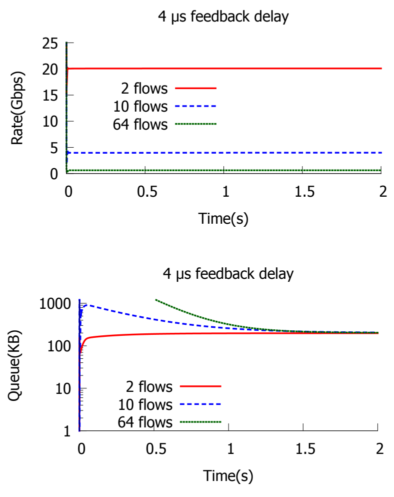


**Figure 18: DCQCN with PI controller**

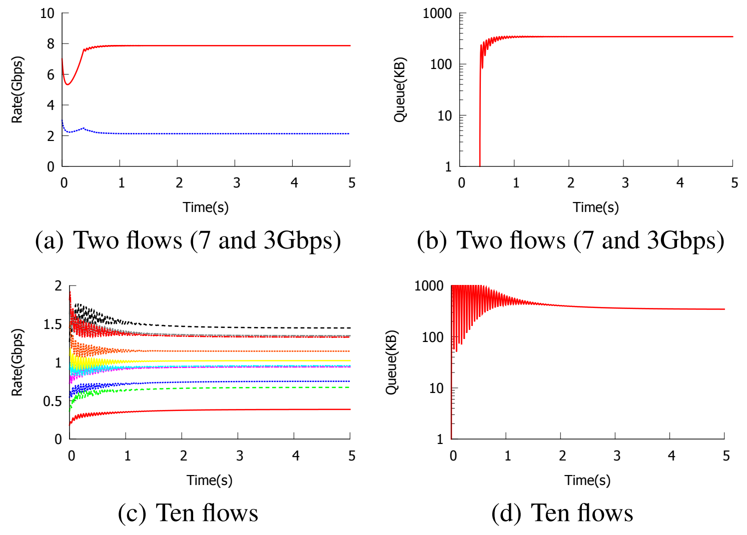


**Figure 19: PI controller to stabilize TIMELY**

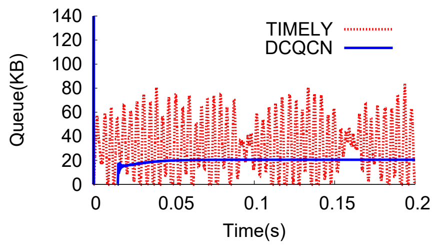


**Figure 20: Protocol stability with random feedback delay up to 100 microseconds. DCQCN is more resilient to jitter than TIMELY.**

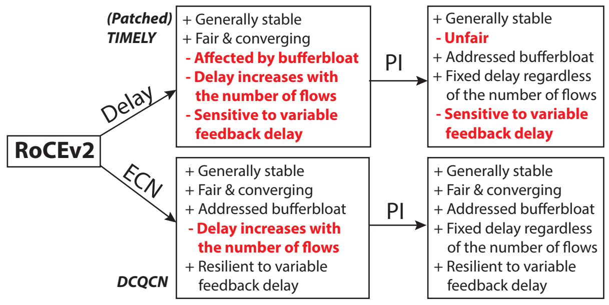

(Patched) TIMELY
RoCEv2
Delay
ECN
DCQCN
+ Generally stable
+ Fair & converging
- Affected by bufferbloat
- Delay increases with the number of flows
- Sensitive to variable feedback delay
PI
+ Generally stable
- Unfair
+ Addressed bufferbloat
+ Fixed delay regardless of the number of flows
- Sensitive to variable feedback delay
+ Generally stable
+ Fair & converging
+ Addressed bufferbloat
- Delay increases with the number of flows
+ Resilient to variable feedback delay
PI
+ Generally stable
+ Fair & converging
+ Addressed bufferbloat
+ Fixed delay regardless of the number of flows
+ Resilient to variable feedback delay

**Figure 21: The design choices and desirable properties.**

**ECN marking is resilient to variable feedback transmission delay:** Last but not the least, ECN marking is more resilient to the delay jitter during feedback transmission. After being generated at the bottleneck in the network, the congestion signal must be conveyed back to the sender. Any hardware jitter or additional congestion in this process can delay the arrival of ECN signal, or interfere round-trip delay measurement. Although DCQCN and TIMELY both attempt to mitigate this artifact, *e.g.,* by prioritizing feedback packets, the hardware jitter or feedback congestion on backward path cannot be completely eliminated in practice.

ECN marking is more resilient to this problem, because the queue length measurement at the bottleneck, and hence the feedback signal is not affected, it is just delayed (more). However, the variable delay of feedback directly injects noise in the TIMELY *feedback'* signal itself. This compounds the problem of congestion or variable delay on the reverse path: for delay based schemes you have delayed *and* noisy feedback, whereas for ECN based schemes you only have delayed feedback. Our simulation confirms this hypothesis. In Figure 20, we inject uniformed random jitter to the feedback delay of DCQCN ($\tau^*$) and TIMELY ($\tau'$) models. With jitter of [0,100$\mu s$], TIMELY becomes unstable compared to the same scenario without the jitter (Figure 12(a)). In contrast, the same level of jitter does not impact DCQCN stability.

## 5.3 Summary

Based on the above three factors, *i.e.,* the faster feedback to the sources, the ability to simultaneously achieve fairness and bounded delay point, and the resilience to variable delay of congestion feedback transmission, we argue in favor of using ECN instead of delay for congestion control in a data center environment. We illustrate this in Figure 21.

**Practical concerns:** ECN can require creating per-flow state on the receiver, if ECN marks must be aggregated. DCQCN [31] does this, since RoCEv2 does not send per-packet ACKs for efficiency reasons. No matter which signal is used, the sender also needs to maintain per-flow state. This may not be scalable since RoCEv2 is implemented on the NIC. Detailed discussion of these issues is outside the scope of this paper.

While PI is not implemented in today's commodity switches, as shown in [14] it is a lightweight mechanism that requires less or comparable computational power as RED, which is supported by all modern switches. A variant of the PI controller (PIE) is being used to solve bufferbloat [23, 22] , and is part of DOCSIS 3.1 standard.

## 6. RELATED WORK

There is a vast amount of literature on congestion signals (drop, ECN, delay), congestion control algorithms and their analysis. See [27] for a succinct overview. Below, we discuss only a few representative papers.

In [3] and [4], Alizadeh et.al. analyze the DCTCP [2] and QCN [16] protocols that DCQCN is derived from. These papers served as useful guideposts during for our work.

Fluid model analysis of TCP under the RED AQM controller, and subsequent development of the PI controller was reported in [20, 14]. Our exploration of the PI controller for DCQCN and TIMELY is guided by results in [14].

A number of congestion control protocols where the bottleneck switch or a central controller plays a more active role have been proposed. For example, RCP [9] and XCP [19] require the switches to send more detailed feedback, while proposals like [29, 30, 24] use an omniscient central controller for fine grain scheduling and pFabric [5] is a timeout based congestion control that requires switches to sort packets. Comparison of these protocols to ECN and delay-based protocols is outside the scope of this paper.

## 7. CONCLUSION AND FUTURE WORK

We analyzed the behavior of two recently proposed congestion control protocols for data center networks; namely DCQCN (ECN based) and TIMELY (delay based). Using fluid models and control theoretic analysis we derived stability regions for DCQCN, which demonstrated a somewhat odd non-monotonic behavior of stability with respect to the number of contending flows. We verified this behavior via packet level simulations. We showed that DCQCN converges to a unique fixed point exponentially. In performing similar analysis for TIMELY, we discovered that as proposed the TIMELY protocol has infinite fixed points which could lead to unpredictable behavior and unbounded unfairness. We provide a simple fix to TIMELY to remedy this problem. The modified protocol is stable, and converges quickly.

However, for both protocols, the operating queue length grows with the number of contending flows, which can introduce significant latency. Using a PI controller on the switch to mark packets, we can guarantee bounded delay and fairness for DCQCN. However we demonstrate and prove a fundamental uncertainty result for delay-based protocols: if you use delay as the only feedback signal for congestion control, then you can either guarantee fairness or a fixed, bounded delay, but not both simultaneously. Based on this reason, the fact that ECN marking process on modern shared-buffer switches effectively excludes queuing delay from feedback loop, and that ECN is more robust to jitter in the feedback, we conclude that ECN is a better congestion signal in data center environment.

**Future work:** we are doing a full exploration of PI like controllers for congestion control protocols of RDMA in the data center, including a hardware implementation. Our analysis also suggests that DCQCN can be simplified considerably, to remove strange artifacts like the non-monotonic stability behavior.

We also plan to analyze problems that are not covered in the paper due to time and space limit. These include multiple bottleneck scenario, larger and realistic topolgoy and workload, and the impact of PFC-induced PAUSES on the two protocols. To capture these complicated behaviors, we will need to develop more analysis tools and improve the performance of DCQCN/TIMELY NS3 simulator [1].

## 8. ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers and our shepherd, Michael Schapira, for their insightful feedback. Vishal Misra was supported in part by NSF Grant CNS-1618911. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF.

## 9. REFERENCES

[1] https://github.com/bobzhuyb/ns3-rdma.

[2] M. Alizadeh, A. Greenberg, D. Maltz, J. Padhye, P. Patel, B. Prabhakar, S. Sengupta, and M. Sridharan. Data Center TCP (DCTCP). In *SIGCOMM*, 2010.

[3] M. Alizadeh, A. Javanmard, and B. Prabhakar. Analysis of DCTCP: Stability, convergence and fairness. In *SIGMETRICS*, 2011.

[4] M. Alizadeh, A. Kabbani, B. Atikoglu, and B. Prabhakar. Stability analysis of QCN: the averaging principle. In *SIGMETRICS*, 2011.

[5] M. Alizadeh, S. Yang, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. Deconstructing datacenter packet transport. In *Proceedings of the 11th ACM Workshop on hot topics in networks*, pages 133–138. ACM, 2012.

[6] S. Athuraliya, D. E. Lapsley, and S. H. Low. An enhanced random early marking algorithm for internet flow control. In *Proceedings of IEEE INFOCOM 2000*, pages 1425–1434, 2000.

[7] A. Dragojevic, D. Narayanan, O. Hodson, and M. Castro. FaRM: Fast remote memory. In *NSDI*, 2014.

[8] N. Dukkipati. Rate control protocol (RCP): Congestion control to make flows complete quickly. In *PhD diss., Stanford University*, 2007.

[9] N. Dukkipati, N. McKeown, and A. G. Fraser. Rcp-ac: Congestion control to make flows complete quickly in any environment. In *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, pages 1–5. IEEE, 2006.

[10] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1:397–413, 1993.

[11] J. Gettys and K. Nichols. Bufferbloat: Dark buffers in the internet. *Queue*, 9(11):40, 2011.

[12] M. Ghobadi, R. Mahajan, A. Phanishayee, J. Kulkarni, G. Ranade, N. Devanur, P.-A. Blanche, H. Rastegarfar, M. Glick, and D. Kilper. Projector: Agile

reconfigurable data center interconnect. In *sigcomm*, 2016.

[13] F. Golnarghi and B. C. Kuo. *Automatic control systems*. Wiely, 2009.

[14] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. On designing improved controllers for aqm routers supporting tcp flows. In *INFOCOM*, 2001.

[15] C. Hollot, V. Misra, D. Towsley, and W.-B. Gong. Analysis and design of controllers for aqm routers supporting tcp flows. *IEEE Transactions on Automatic Control*, 2002.

[16] IEEE. 802.11Qau. Congestion notification, 2010.

[17] IEEE. 802.11Qbb. Priority based flow control, 2011.

[18] Infiniband Trade Association. Supplement to InfiniBand architecture specification volume 1 release 1.2.2 annex A17: RoCEv2 (IP routable RoCE), 2014.

[19] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. *ACM SIGCOMM Computer Communication Review*, 32(4):89–102, 2002.

[20] V. Misra, W.-B. Gong, and D. Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 151–160. ACM, 2000.

[21] R. Mitta, E. Blem, N. Dukkipati, T. Lam, A. Vahdat, Y. Wang, H. Wassel, D. Wetherall, D. Zats, and M. Ghobadi. TIMELY: RTT-based congestion control for the datacenter. In *SIGCOMM*, 2015.

[22] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. https://tools.ietf.org/html/draft-pan-tsvwg-pie-00.

[23] R. Pan, P. Natarajan, C. Piglione, M. S. Prabhu, V. Subramanian, F. Baker, and B. VerSteeg. Pie: A lightweight control scheme to address the bufferbloat problem. In *HPSR*, pages 148–155. IEEE, 2013.

[24] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A centralized zero-queue datacenter network. In *Proceedings of the 2014 ACM conference on SIGCOMM*, pages 307–318. ACM, 2014.

[25] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN). RFC 3168.

[26] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. Low extra delay background transport (ledbat). Technical report, 2012.

[27] R. Srikant. *The mathematics of Internet congestion control*. Springer Science & Business Media, 2012.

[28] B. Stephens, A. Cox, A. Singla, J. Carter, C. Dixon, and W. Felter. Practical DCB for improved data center networks. In *INFOCOMM*, 2014.

[29] B. C. Vattikonda, G. Porter, A. Vahdat, and A. C. Snoeren. Practical tdma for datacenter ethernet. In *EuroSys*, pages 225–238. ACM, 2012.

[30] C. Wilson, H. Ballani, T. Karagiannis, and A. Rowstron. Better never than late: Meeting deadlines in datacenter networks. In *SIGCOMM*, 2011.

[31] Y. Zhu, H. Eran, D. Firestone, C. Guo, M. Lipshteyn, Y. Liron, J. Padhye, S. Raindel, M. H. Yahia, and M. Zhang. Congestion Control for Large-Scale RDMA Deployments. In *SIGCOMM*, 2015.

[32] The NS3 Simulator. https://www.nsnam.org/.

## APPENDIX

## A. DERIVING DCQCN CHARACTERISTIC EQUATION

We derive DCQCN characteristic equation by linearizing the system and Laplace transform.

**Linearization.** Below we denote $\delta R_C(t) = R_C(t) - R_C^*$, $\delta R_C(t) = R_C(t) - R_C^*$, $\delta p(t) = p(t) - p^*$, $\delta\alpha(t) = \alpha(t) - \alpha^*$, and $A = \left(\frac{1}{B} + \frac{1}{TR_C^*}\right)$. We again use Taylor series to simplify the expressions of $a, b, c, d, e$ to handle the exponential forms like $(1-p)^x$. Due to lack of space, here we only just show the linearized expression for $\frac{d\delta R_C}{dt}$:

$$\begin{aligned}
&\tfrac{d\delta R_C}{dt} = -\tfrac{1}{2}(R_C^*)^2\alpha^*\delta p - \tfrac{1}{2}p^*R_C^*\alpha^*\delta R_C \\
&-\tfrac{1}{2}p^*R_C^*\alpha^*\delta R_C - \tfrac{1}{2}p^*(R_C^*)^2\delta\alpha \\
&+\tfrac{A}{2}\left(R_C^*\delta R_T - R_C^*\delta R_C + R_T^*\delta R_C - R_C^*\delta R_C\right) \\
&-\left(\tfrac{1}{2}+\tfrac{A}{4}\right)\left(p^*R_C^*\delta R_T - p^*R_C^*\delta R_C + p^*R_T^*\delta R_C\right) \\
&-\left(\tfrac{1}{2}+\tfrac{A}{4}\right)\left(p^*R_C^*\delta R_C - R_C^*R_T^*\delta p + (R_C^*)^2\delta p\right)
\end{aligned} \tag{33}$$

**Laplace transform.** We get the Laplace transform of the above linearized model:

$$\begin{aligned}
&sR_C(s) - \delta R_C(0) = \\
&\left(-\tfrac{1}{2}(R_C^*)^2\alpha^* - \left(\tfrac{1}{2}+\tfrac{A}{4}\right)R_C^*R_T^* + \left(\tfrac{1}{2}+\tfrac{A}{4}\right)(R_C^*)^2\right)e^{-s\tau*}p(s) \\
&+\left(-\tfrac{1}{2}p^*R_C^*\alpha^* - \tfrac{A}{2}R_C^* + \left(\tfrac{1}{2}+\tfrac{A}{4}\right)p^*R_C^*\right)e^{-s\tau*}R_C(s) \\
&+\left(-\tfrac{1}{2}p^*R_C^*\alpha^* - \tfrac{A}{2}R_C^* + \tfrac{A}{2}R_T^*\right)R_C(s) \\
&+\left(\left(\tfrac{1}{2}+\tfrac{A}{4}\right)p^*R_C^* - \left(\tfrac{1}{2}+\tfrac{A}{4}\right)p^*R_T^*\right)R_C(s) \\
&-\tfrac{1}{2}p^*(R_C^*)^2\alpha(s) \\
&+\left(\tfrac{A}{2}R_C^* - \left(\tfrac{1}{2}+\tfrac{A}{4}\right)p^*R_C^*\right)R_T(s)
\end{aligned} \tag{34}$$

With Laplace transform of the other equations, we can use $R_C(s)$ to express $R_T(s)$, $p(s)$ and $\alpha(s)$. We then derive the characteristic equation of $R_C(s)$. Finally we compute the phase margin values of the characteristic equation with different parameters, and show the results in Section 3.2.

## B. PROOF OF DCQCN CONVERGENCE

Below we provide the additional details of our proof for Theorem 2. Figure 22 illustrates the variables we use.

**Part I: The analysis of $R_T$.** During the consecutive additive rate increase, *i.e.*, $\forall t \in (T_k+1, T_{k+1}]$, $R_C^{(i)}$ and $R_T^{(i)}$ have following relationship according to DCQCN's definition:

$$R_C^{(i)}(t+1) = \frac{1}{2}\left(R_C^{(i)}(t) + R_T^{(i)}(t+1)\right) \tag{35}$$

$$R_T^{(i)}(t+1) = R_T^{(i)}(t) + R_{AI} \tag{36}$$

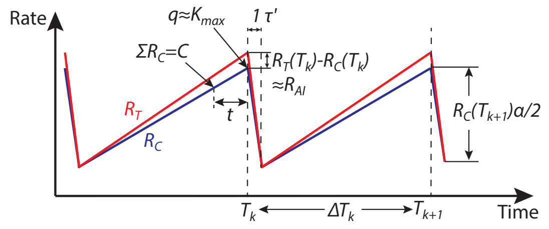


**Figure 22: A more detailed view of DCQCN's AIMD-style updates of flow rate.**

By (35)-$\frac{1}{2}$×(36), we get:

$$\begin{aligned}&R_C^{(i)}(T_{k+1}) - R_T^{(i)}(T_{k+1}) + R_{AI} \\ &= \tfrac{1}{2}\left(R_C^{(i)}(T_{k+1}-1) - R_T^{(i)}(T_{k+1}-1) + R_{AI}\right) \\ &= ... = \left(\tfrac{1}{2}\right)^{\Delta T_k - 1}\left(R_C^{(i)}(T_k+1) - R_T^{(i)}(T_k+1) + R_{AI}\right)\end{aligned} \tag{37}$$

From this, we know that during a consecutive additive rate increase phase, $R_T^{(i)} - R_C^{(i)}$ will converge towards $R_{AI}$ exponentially. In addition, at the beginning of the phase, $R_C^{(i)}(T_k + 1) = R_T^{(i)}(T_k)$ (Figure 6). Therefore:

$$R_T^{(i)}(T_k) = R_C^{(i)}(T_k) + \left(1 - (\frac{1}{2})^{\Delta T_k - 1}\right) R_{AI}, \forall k = 1, 2, ... \tag{38}$$

**Part II: The lower bound of $\alpha$.** To prove Equation 19, we first need to estimate $\Delta T_k$ using $\alpha$. In the period of $\Delta T_k$, after the first time unit of rate decrease, the aggregated flow rates will climb back to $R_T(T_{k+1})$ by $NR_{AI}$ every time unit. Thus we have:

$$\Delta T_k = 1 + \frac{\sum_{i=1}^{N}\left(R_T^{(i)}(T_{k+1}) - \left(1 - \alpha^{(i)}(T_k)/2\right) R_C^{(i)}(T_k)\right)}{NR_{AI}} \tag{39}$$

Suppose $\alpha^{(i)}(T_k)$ already converged to the same value $\alpha(T_k)$, as guaranteed by Equation 17. We simplify it as:

$$\begin{aligned}\Delta T_k &= 1 + \tfrac{\sum_{i=1}^{N} R_T^{(i)}(T_{k+1}) - (1-\alpha(T_k)/2)\sum_{i=1}^{N} R_C^{(i)}(T_k)}{NR_{AI}} \\ &\approx 1 + \tfrac{(C+tNR_{AI}+NR_{AI}) - (1-\alpha(T_k)/2)(C+tNR_{AI})}{NR_{AI}} \\ &= 2 + \left(\tfrac{t}{2} + \tfrac{C}{2NR_{AI}}\right)\alpha(T_k)\end{aligned} \tag{40}$$

Where $t$ is the time it takes for the flows to build up queue and get packets ECN-marked, after the aggregated flow rates exceed link capacity $C$, as shown in Figure 6. We can estimate $t$ by the queue being built up:

$$\begin{aligned}&N\tau'\left(R_{AI} + 2R_{AI} + ... + tR_{AI}\right) = Q_{ECN} \le K_{\max} \\ &\Rightarrow t \le \left(-1 + \sqrt{1 + \tfrac{8K_{\max}}{NR_{AI}\tau'}}\right)/2\end{aligned} \tag{41}$$

Now, we prove Equation 19, where $\alpha^*$ is the solution of the following:

$$\alpha^* = (1-g)^{\Delta T^*}\left((1-g)\alpha^* + g\right) \tag{42}$$

Once Equation 19 is proved, $\alpha(T_k)$ has a non-zero lower bound, $R_C$ will converge exponentially. We prove this by mathematical induction. The initial value of $\alpha$ is 1, as defined by DCQCN. So, $\alpha(T_0) > \alpha^* > 0$. Now assuming $\alpha(T_k) > \alpha^* > 0$, we prove $\alpha(T_k) > \alpha(T_{k+1}) > \alpha^*$. We define $f(\alpha)$ as the RHS of Equation 16:

$$f(\alpha) = (1-g)^{2+\left(\frac{t}{2}+\frac{C}{2NR_{AI}}\right)\alpha}\left((1-g)\alpha + g\right) \tag{43}$$

By analyzing the derivative of $f(\alpha)$, it is not hard to see that with common parameter settings, $f(\alpha)$ is monotonically increasing. Therefore,

$$\alpha(T_{k+1}) = f\left(\alpha(T_k)\right) > f\left(\alpha^*\right) = \alpha^* \tag{44}$$

In addition, because $\alpha(T_k) > \alpha^* \Rightarrow \Delta T_k > \Delta T^*$, $\alpha^*$ satisfies:

$$\alpha^* > (1-g)^{\Delta T_k}\left((1-g)\alpha^* + g\right) \tag{45}$$

Subtract this from Equation 16, we see $\alpha(T_k)$ is exponentially converging towards $\alpha^*$:

$$\alpha(T_{k+1}) - \alpha^* < (1-g)^{\Delta T_n}\left(\alpha(T_k) - \alpha^*\right) \tag{46}$$

Equation 44 and 46 lead to $\alpha^* < \alpha(T_{k+1}) < \alpha(T_k)$. □