POWERTCP: Pushing the Performance Limits of Datacenter Networks*

Vamsi Addanki
TU Berlin
University of Vienna

Oliver Michel
Princeton University
University of Vienna

Stefan Schmid
TU Berlin
University of Vienna

Abstract

Increasingly stringent throughput and latency requirements in datacenter networks demand fast and accurate congestion control. We observe that the reaction time and accuracy of existing datacenter congestion control schemes are inherently limited. They either rely only on explicit feedback about the network state (e.g., queue lengths in DCTCP) or only on variations of state (e.g., RTT gradient in TIMELY). To overcome these limitations, we propose a novel congestion control algorithm, POWERTCP, which achieves much more fine-grained congestion control by adapting to the bandwidth-window product (henceforth called power). POWERTCP leverages in-band network telemetry to react to changes in the network instantaneously without loss of throughput and while keeping queues short. Due to its fast reaction time, our algorithm is particularly well-suited for dynamic network environments and bursty traffic patterns. We show analytically and empirically that POWERTCP can significantly outperform the state-of-the-art in both traditional datacenter topologies and emerging reconfigurable datacenters where frequent bandwidth changes make congestion control challenging. In traditional datacenter networks, POWERTCP reduces tail flow completion times of short flows by 80% compared to DCQCN and TIMELY, and by 33% compared to HPCC even at 60% network load. In reconfigurable datacenters, POWERTCP achieves 85% circuit utilization without incurring additional latency and cuts tail latency by at least 2x compared to existing approaches.

1 Introduction

The performance of more and more cloud-based applications critically depends on the underlying network, requiring datacenter networks (DCNs) to provide extremely low latency and high bandwidth. For example, in distributed machine learning applications that periodically require large data transfers, the network is increasingly becoming a bottleneck [36]. Similarly, stringent performance requirements are introduced by today's trend of resource disaggregation in datacenters where fast access to remote resources (e.g., GPUs or memory) is pivotal for the overall system performance [36]. Building systems with strict performance requirements is especially challenging under bursty traffic patterns as they are commonly observed in datacenter networks [12, 16, 47, 53, 55].

These requirements introduce the need for fast and accurate network resource management algorithms that optimally utilize the available bandwidth while minimizing packet latencies and flow completion times. Congestion control (CC) plays an important role in this context being ''a key enabler (or limiter) of system performance in the datacenter'' [34]. In fact, fast reacting congestion control is not only essential to efficiently adapt to bursty traffic [29, 48], but is also becoming increasingly important in the context of emerging reconfigurable datacenter networks (RDCNs) [13, 14, 20, 33, 38, 39, 50]. In these networks, a congestion control algorithm must be able to quickly ramp up its sending rate when high-bandwidth circuits become available [43].

Traditional congestion control in datacenters revolves around a bottleneck link model: the control action is related to the state i.e., queue length at the bottleneck link. A common goal is to efficiently control queue buildup while achieving high throughput. Existing algorithms can be broadly classified into two types based on the feedback that they react to. In the following, we will use an analogy to electrical circuits[1] to describe these two types. The first category of algorithms react to the absolute network state, such as the queue length or the RTT: a function of network ''effort'' or **voltage** defined as the sum of the bandwidth-delay product and in-network queuing. The second category of algorithms rather react to variations, such as the change of RTT. Since these changes are related to the network ''flow'', we say that these approaches depend on the **current** defined as the total transmission rate. We tabulate our analogy and corresponding network quantities in Table 1. According to this classification, we call congestion control protocols such as CUBIC [21], DCTCP [7], or Vegas [15] **voltage-based CC** algorithms as

*Research was conducted at the University of Vienna during 2020-21.

[1] This analogy is inspired from S. Keshav's lecture series based on mathematical foundations of computer networking [31]. We emphasize that our power analogy is meant for the networking context considered in this paper and it should not be applied to other domains of science.

Quantity	Analogy
Total transmission rate (network flow)	Current (λ)
BDP + buffered bytes (network effort)	Voltage (ν)
Current × Voltage	Power (Γ)

Table 1: Analogy between metrics in networks and in electrical circuits. Note that the network here is the ''pipe'' seen by a flow and not the whole network.

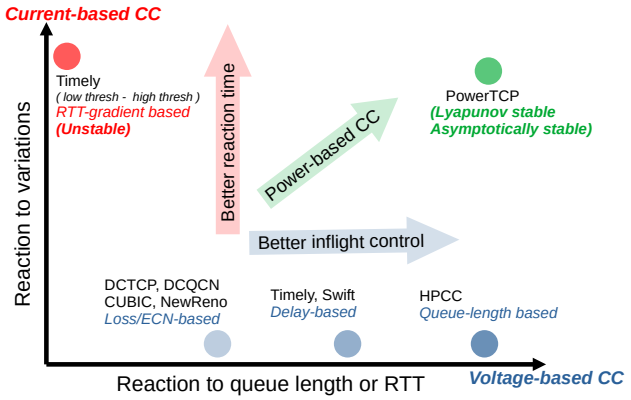


Figure 1: Existing congestion control algorithms are fundamentally limited to a single dimension in their window (or rate) update decisions and are unable to distinguish between two scenarios across multiple dimensions.

they react to absolute properties such as the bottleneck queue length, delay, Explicit Congestion Notification (ECN), or loss. Recent proposals such as TIMELY [41] are **current-based CC** algorithms as they react to the variations, such as the RTT-gradient. In conclusion, we find that existing congestion control algorithms are fundamentally limited to one of the two dimensions (voltage or current) in the way they update the congestion window.

We argue that the input to a congestion control algorithm should rather be a function of the two-dimensional state of the network (i.e., both voltage and current) to allow for more informed and accurate reaction, improving performance and stability. In our work, we show that there exists an accurate relationship between the optimal adjustment of the congestion window, the network voltage and the network current. We analytically show that the optimal window adjustment depends on the product of network voltage and network current. We call this product **network power**: current × voltage, a function of both queue lengths and queue dynamics.

Figure 1 illustrates our classification. Existing protocols depend on a single dimension, voltage or current. This can result in imprecise congestion control as the protocol is unable to distinguish between fundamentally different scenarios, and, as a result, either reacts too slowly or overreacts, both impeding performance. Accounting for both voltage and current, i.e., power, balances accurate inflight control and fast reaction, effectively providing the best of both worlds.

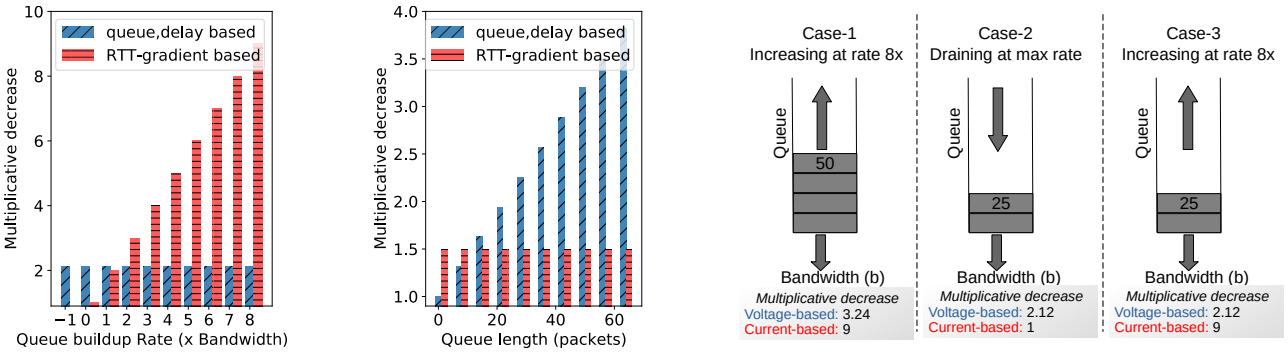
In this paper we present PowerTCP, a novel *power*-based congestion control algorithm that accurately captures both *voltage* and *current* dimensions for every control action using measurements taken within the network and propagated through in-band network telemetry (INT). PowerTCP is able to utilize available bandwidth within one or two RTTs while being stable, maintaining low queue lengths, and resolving congestion rapidly. Furthermore, we show that PowerTCP is Lyapunov-stable, as well as asymptotically stable and has a convergence time as low as five update intervals (Appendix A). This makes PowerTCP highly suitable for today's datacenter networks and dynamic network environments such as in reconfigurable datacenters.

PowerTCP leverages in-network measurements at programmable switches to accurately obtain the bottleneck link state. Our switch component is lightweight and the required INT header fields are standard in the literature [36]. We also discuss an approximation of PowerTCP for use with non-programmable, legacy switches.

To evaluate PowerTCP, we focus on a deployment scenario in the context of RDMA networks where the CC algorithm is implemented on a NIC. Our results from large-scale simulations show that PowerTCP reduces the 99.9-percentile short flow completion times by 80% compared to DCQCN [56] and by 33% compared to the state-of-the-art low-latency protocol HPCC [36]. We show that PowerTCP maintains near-zero queue lengths without affecting throughput or incurring long flow completion times even at 80% load. As a case study, we explore the benefits of PowerTCP in reconfigurable datacenter networks where it achieves $80-85\%$ circuit utilization and reduces tail latency by at least $2\times$ compared to the state-of-the-art [43]. Finally, as a proof-of-concept, we implemented PowerTCP in the Linux kernel and the telemetry component on an Intel Tofino programmable line-rate switch using P4 [18].

In summary, our key contributions in this paper are:

- We reveal the shortcomings of existing congestion control approaches which either only react to the current state or the dynamics of the network, and introduce the notion of *power* to account for both.
- PowerTCP, a power-based approach to congestion control at the end-host which reacts faster to changes in the network such as an arrival of burst, fluctuations in available bandwidth etc.,
- An evaluation of the benefits of PowerTCP in traditional DCNs and RDCNs.
- As a contribution to the research community and to facilitate future work, all our artefacts have been made publicly available at: https://powertcp.self-adjusting.net.



(a) Voltage-based CC is oblivious to queue buildup rate.

(b) Current-based CC is oblivious to queue lengths.

(c) Voltage-based CC cannot differentiate case-2 vs case-3; whereas current-based CC cannot differentiate case-1 vs case-3.

Figure 2: Existing CC schemes, classified as voltage and current-based, are orthogonal in their response to queue length and queue buildup rate.

2 Motivation

We first provide a more detailed motivation of our work by highlighting the benefits and drawbacks of existing congestion control approaches. In the following, **voltage-based CC** refers to the class of end-host congestion control algorithms that react to the state of the network in absolute values related to the bandwidth-delay product, such as bottleneck queue length, delay, loss, or ECN; **current-based CC** refers to the class of algorithms that react to changes in the state, such as the RTT-gradient. Voltage-based CC algorithms are likely to exhibit better stability but are fundamentally limited in their reaction time. Current-based CC algorithms detect congestion faster but ensuring stability may be more challenging. Indeed, TIMELY [41], a current-based CC, deployed at Google datacenters, turned out to be unstable [57] and evolved to SWIFT [34], a voltage-based CC.

Orthogonal to our approach, receiver-driven transport protocols [22, 26, 42] have been proposed which show significant performance improvements. A receiver-driven transport approach relies on the assumption that datacenter networks are well-provisioned and claims that congestion control is unnecessary; for example ‘‘NDP performs no congestion control whatsoever in a Clos topology’’ [22]. The key difference is that receiver-driven approaches take feedback from the ToR downlink at the receiver which can only identify congestion at the last hop, whereas sender-based approaches rely on a variety of feedback signals to identify congestion anywhere along the path. In this paper, we focus on the sender-based congestion control approach which can in principle handle congestion anywhere along the round-trip path between a sender and a receiver, even in oversubscribed datacenters.

To take a leap forward and design fine-grained datacenter congestion control algorithms, we present an analytical approach and study the fundamental problems faced by existing algorithms. We first formally express the desirable properties of a datacenter congestion control law (§2.1) and then analytically identify the drawbacks of existing control laws (§2.2). Finally, we discuss the lessons learned and formulate our design goals (§2.3).

2.1 Desirable Control Law Properties

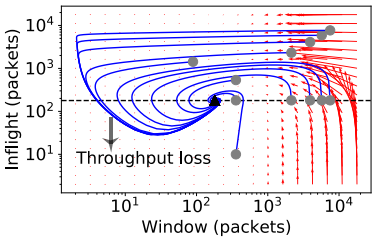
Among various desired properties of datacenter congestion control, high throughput and low tail latency are most important [7, 36, 41] with fairness and stability being essential as well [54, 57]. Achieving these properties simultaneously can be challenging. For example, to realize high throughput, we may aim to keep the queue length at the bottleneck link large; however, this may increase latency. Thus, an ideal CC algorithm must be capable of maintaining near-zero queue lengths, achieving both high throughput and low latency. It must further minimize throughput loss and latency penalty caused by perturbations, such as bursty traffic.

In order to formalize our requirements, we consider a single-bottleneck link model widely used in the literature [24, 40, 54, 57]. Specifically, we assume that all senders use the same protocol, transmit long flows[2] sharing a common bottleneck link with bandwidth b, and have a base round trip time τ (excluding queuing delays). In this model, equilibrium is a state reached when the window size and bottleneck queue length stabilize. We now formally express the desired equilibrium state that captures our performance requirements in terms of the sum of window sizes of all flows (aggregate window size) $w(t)$, bandwidth delay product $b \cdot \tau$, and bottleneck queue length $q(t)$:

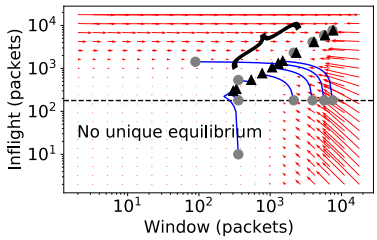
$$0 < q(t) < \varepsilon \qquad (1)$$
$$b \cdot \tau \leq w(t) < b \cdot \tau + \varepsilon$$
$$\dot{q}(t) = 0;\ \dot{w}(t) = 0$$

where ε is a positive integer. First, this captures the requirement for high throughput i.e., when $w(t) > b \cdot \tau$ and $q(t) > 0$, the number of inflight bytes are greater than the bandwidth-delay product (BDP) and the queue length is greater than zero.

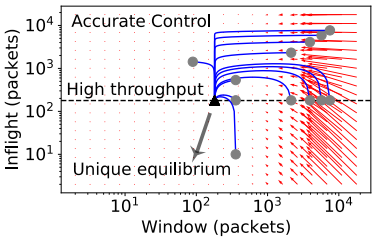
[2] Note that, although most DC flows are short flows, most DC traffic volume (bytes) is from long flows [7, 9].



(a) Voltage-based CC (RTT or queue length) exhibits equilibrium properties but has an imprecise reaction leading to throughput loss.



(b) Current-based CC (RTT-gradient) reacts faster but has no unique equilibrium point, and is thereby unable to stabilize queue lengths.



(c) PowerTCP, a power-based CC, exhibits equilibrium properties and has a precise reaction to perturbations.

Figure 3: Phase plots showing the trajectories of existing schemes and our approach PowerTCP from different initial states (circles) to equilibrium (triangles). At each point on the plane, arrows show the direction in which the system moves. An example is depicted with bottleneck link bandwidth 100Gbps and a base RTT of 20μs. BDP is shown by a horizontal dotted line and any trajectory going below this line indicates throughput loss.

Second, from $w(t) < b \cdot \tau + \varepsilon$ and $q(t) < \varepsilon$, the queue length is at most ε, thereby achieving low latency. Finally, for the system to stabilize, we need that $\dot{q}(t) = 0$ and $\dot{w}(t) = 0$.

As simple as these requirements are, it is challenging to control the aggregate window size $w(t)$ while CC operates per flow. In addition to the equilibrium state requirement, we need fast response to perturbations. The response must minimize the distance from the equilibrium i.e., minimize the latency or throughput penalty caused by a perturbation (e.g., incast or changes in available bandwidth).

In this work, we ask two fundamental questions:

***(Q1)* Equilibrium point:** Do existing algorithms satisfy the equilibrium state in Eq. 1 for the aggregate window size? In addition to the equilibrium behavior, we are also interested in the reaction to a perturbation.

***(Q2)* Response to perturbation:** What is the trajectory followed after a perturbation, i.e., the dynamics of the bottleneck queue as well as the TCP window sizes, from an initial point to the equilibrium point?

2.2 Drawbacks of Existing Control Laws

We now aim to analytically answer our questions above and shed light on the inefficiencies of existing protocols, both voltage-based and current-based. We begin by simplifying the congestion avoidance model of existing CC approaches we are interested in, specifically delay, queue length, and RTT-gradient based CC approaches as follows:

$$w_i(t+\delta t) = \gamma \cdot \left(w_i(t) \cdot \frac{e}{f(t)} + \beta \right) + (1-\gamma) \cdot w_i(t) \qquad (2)$$

Here w_i is the window of a flow i, β is the additive increase term, e is the equilibrium point that the algorithm is expected to reach, $f(t)$ is the measured feedback and γ is the exponential moving average parameter. A queue length-based CC [36] sets the desired equilibrium point e as $b \cdot \tau$ (BDP) and the feedback $f(t)$ as the sum of bottleneck queue length and BDP i.e., voltage (ν). A delay-based CC [34] sets e to τ (base RTT) and the feedback $f(t)$ as RTT which is the sum of queuing delay and base RTT i.e., $\frac{\text{voltage}}{\text{bandwidth}}$ ($\frac{\nu}{b}$). Similarly, the RTT-gradient approach [41] sets e to 1 and the feedback $f(t)$ as one plus RTT-gradient i.e., $\frac{\text{current}}{\text{bandwidth}}$ ($\frac{\lambda}{b}$). In Appendix B, we further justify how Eq. 2 captures existing control laws[3]. Note that our simplified model does not capture loss/ECN-based CC algorithms; however, there exists rich literature on the analysis of loss/ECN-based CC algorithms [24, 37] including DCTCP [7, 8]. We now use Euler's first order approximation to obtain the window dynamics as follows:

$$\dot{w}_i(t) = \frac{\gamma}{\delta t} \cdot \left(w_i(t) \cdot \frac{e}{f(t)} - w_i(t) + \beta \right) \qquad (3)$$

Each flow i has a sending rate λ_i and hence the bottleneck queue experiences an aggregate arrival rate of λ. In our analogy, λ is the network current. We additionally use the traditional model of queue length dynamics which is independent of the control law [24, 40]:

$$\dot{q}(t) = \begin{cases} \lambda(t - t^f) - \mu(t) & q(t) > 0 \\ 0 & otherwise \end{cases} \qquad (4)$$

where $\lambda(t) = \frac{w(t)}{\theta(t)}$. An equilibrium point is the window size w_e and queue length q_e that satisfies $\dot{w}(t) = 0$ and $\dot{q}(t) = 0$.

We are now ready to answer the questions raised.

Equilibrium point: It is well-known from literature that loss/ECN-based schemes operate by maintaining a standing queue [8, 24, 27]. For example, TCP NewReno flows fill the queue to maximum (say q_{max}) and then react by reducing windows by half. Consequently, the bottleneck queue-length oscillates between q_{max} and $q_{max} - b \cdot \tau$ or zero if $q_{max} < b \cdot \tau$. DCTCP flows oscillate around the marking threshold $K > \frac{b \cdot \tau}{7}$

[3] TIMELY, for example, is rate-based while our simplification is window-based. However, window and rate are interchangeable for update calculations.

which depends on BDP [7]. This does not satisfy our stringent requirement in Eq. 1. While ECN-based schemes reduce the amount of standing queue required, we still consider the standing queue which is proportional to bandwidth to be unacceptable given the increasing gap between bandwidth vs switch buffers.

It can be shown that there exists a unique equilibrium point for queue length and delay approaches (voltage-based CC) defined by Eq. 2. However, current-based CC and, in particular, RTT-gradient approaches do not have a unique equilibrium point suggesting a lack of control over queue lengths. Intuitively, RTT-gradient approaches quickly adapt the sending rate to stabilize the RTT-gradient ($\dot{\theta} = \frac{\dot{q}}{b}$) which in turn only stabilizes the queue length gradient $\dot{q}(t)$ but fails to control the absolute value of the queue length. It has indeed been shown that TIMELY, a current-based CC does not have a unique equilibrium [57].

Figure 3 visualizes the system behavior according to the window dynamics in Eq. 3 and the queue dynamics in Eq. 4. In Figure 3a we can see that voltage-based CC eventually reaches a unique equilibrium point. In contrast, in Figure 3b we see that current-based CC reaches different final points for different initial points, indicating that there exists no unique equilibrium point thereby violating the desired equilibrium state properties (Eq. 1). To give more context on this observation, in Figure 2 we show the reactions of different schemes for observed queue lengths and queue buildup rate. In Figure 2b, we can see that current-based CC has the same reaction for different queue lengths but exhibits a proportional reaction to queue buildup rate (Figure 2a); consequently, current-based CC cannot stabilize at a unique equilibrium point. Due to space constraints, we move the detailed proof of equilibrium points to Appendix B.

Takeaway. *While voltage-based CC can in principle meet the desired equilibrium state requirements in Eq. 1, current-based CC cannot.*

Response to perturbation: We observe an orthogonal behavior in the responses of voltage-based CC and current-based CC. In Figure 2b we show that voltage-based CC has a proportional reaction to increased queue lengths but a current-based CC approach has the same response for any queue length. Further in Figure 2a we observe that current-based CC has a proportional reaction to the rate at which queue is building up but a voltage-based CC has the same reaction for any rate of queue build up. This orthogonality in existing schemes often results in scenarios with either insufficient reaction or overreaction. To underline our observation, we use the system of differential equations (Eq. 3 and Eq. 4) to observe the trajectories taken by different control laws after a perturbation. We show the trajectories in Figure 3. Specifically, Figure 3a shows that voltage-based CC (queue length or delay based) eventually reaches a unique equilibrium point but overreacts in the response and losing throughput (window $<$ BDP and $q(t) = 0$) almost for every initial point. In Figure 3b we observe that current-based CC (RTT-gradient) reaches different end points for different initial states and consequently does not have a single equilibrium point. However, we see that the initial response is faster with current-based CC due to their use of RTT-gradient which is arguably a superior signal to detect congestion onset even at low queue lengths.

Takeaway. *Current-based CC is superior in terms of fast reaction but lacks equilibrium state properties while voltage-based CC eventually reaches a unique equilibrium but overreacts in its response for almost any initial state resulting in long trajectories from initial state to equilibrium state.*

2.3 Lessons Learned and Design Goals

From our analysis we derive two key observations. First, both voltage and current-based CC have individual benefits. Particularly, voltage-based CC is desirable for the stringent equilibrium properties we require and current-based CC is desirable for fast reaction. Second, both voltage and current-based CC have drawbacks. On one hand, voltage-based CC is oblivious to congestion onset at low queue lengths and on the other hand current-based CC is oblivious to the absolute value of queue lengths. Moreover, voltage-based CC overreacts when the queue drains essentially losing throughput immediately after.

Based on these observations, our goal is to design a control law that systematically combines both voltage and current for every window update action. Specifically our aim is to design a congestion control algorithm with *(i)* equilibrium properties from Eq. 1 exhibited by voltage-based CC and *(ii)* fast response to perturbation exhibited by current-based CC. The challenges are to avoid inheriting the drawbacks of both types of CC, stability and fairness. However in order to design such a control law we face the following challenges:

- Finding an accurate relationship between window, voltage and current. ▷ Property 1
- Ensuring stability, convergence and fairness. ▷ Theorem 1, 2, 3

3 Power-Based Congestion Control

Reflecting on our observations in §2, we seek to design a congestion control algorithm that systematically reacts to both the absolute value of the bottleneck queue length and its rate of change. Our aim is to address today's datacenter performance requirements in terms of high throughput, low latency, and fast reaction to bursts and bandwidth fluctuations.

3.1 The Notion of Power

To address the challenges faced by prior datacenter congestion control algorithms and to optimize along both dimensions, we introduce the notion of *power* associated with the network pipe. Following the bottleneck link model from literature [24, 40], from Eq. 4 we observe that the window size is indeed related to the product of network voltage and network current which we call *power* (Table 1). This corresponds to the product of *(i)* total sending rate λ (current) and *(ii)* the

sum of BDP plus the accumulated bytes q at the bottleneck link (voltage), formally expressed in Eq. 5.

$$\underbrace{\Gamma(t)}_{\text{power}} = \underbrace{(q(t)+b\cdot\tau)}_{\text{voltage}}\cdot\underbrace{\lambda(t-t^f)}_{\text{current}} \tag{5}$$

Notice that the unit of power is $\frac{bit^2}{second}$. We will show the useful properties of power specifically under congestion. Using Eq. 4, we can rewrite Eq. 5 in terms of queue length gradient $\dot{q}$ and the transmission rate μ as,

$$\Gamma(t) = (q(t)+b\cdot\tau)\cdot(\dot{q}(t)+\mu(t)) \tag{6}$$

We now derive a useful property of power using Eq. 6 and Eq. 4 showing an accurate relationship of power and window.

Property 1 (Relationship of Power and Congestion Window)**.** *Power is the bandwidth-window product*

$$\Gamma(t) = b\cdot w(t-t^f)$$

Note that the property is over the aggregate window size i.e., the sum of window sizes of all flows sharing the common bottleneck. We emphasize that our notion of power is intended for the networking context and cannot be applied to other domains of science. In the following, we outline the benefits of considering the notion of power and how Property 1 can be useful in the context of congestion control.

3.2 Benefits of Power-Based CC

A power-based control law can exploit Property 1 to precisely update per flow window sizes. Accurately controlling aggregate window size is a key challenge for an end-host congestion control algorithm. A power-based CC overcomes this challenge by gaining precise knowledge about the aggregate window size from measured power. First, using power enables the window update action to account for the bottleneck queue lengths as well as the queue build-up rate. As a result, a power-based CC can rapidly detect congestion onset even at very low queue lengths. At the same time, a power-based CC also reacts to the absolute value of queue lengths, effectively dampening perturbations. Second, calculating power at the end-host requires no extra measurement and feedback mechanisms compared to INT based schemes such as HPCC [36].

3.3 The PowerTCP Algorithm

Driven by our observations, we carefully designed our control law based on power, capturing a systematic reaction to voltage (related to bottleneck queue length), as well as to current (related to variations in the bottleneck queue length).

Control law: PowerTCP is a window-based congestion control algorithm and updates its window size upon receipt of an acknowledgment. For a flow i, every window update is based on *(i)* current window size $w_i(t)$, *(ii)* additive increase β, *(iii)* window size at the time of transmission of the acknowledged segment $w_i(t-\theta(t))$, and *(iv)* power measured from the feedback information. We refer the reader to Table 2 for the general notations being used. Formally, PowerTCP's control law can be expressed as

$$w_i(t) \leftarrow \gamma\cdot\left(w_i(t-\theta(t))\cdot\frac{e}{f(t)}+\beta\right)+(1-\gamma)\cdot w_i(t) \tag{7}$$

$$e = b^2\cdot\tau; \quad f(t) = \Gamma(t-\theta(t)+t^f)$$

where $\gamma\in(0,1]$ and β are parameters to the control law. The base round trip time τ must be configured at compile time. If baseRTT is not precisely known, an alternative is to keep track of minimum observed RTT. We first describe how power Γ is computed and then present the pseudocode of PowerTCP in Algorithm 1.

Feedback: PowerTCP's control law is based on power. Note that power (Eq. 5) is only related to variables at the bottleneck link. In order to measure power, we leverage in-band network telemetry. Specifically, the workings of INT and the header fields required are the same as in HPCC (Figure. 4 in [36]). When a TCP sender sends out a packet P into the network, it additionally inserts an INT header INT into the packet. Each switch along the path then pushes metadata containing the egress queue length (*qlen*), timestamp (*ts*), so far transmitted bytes (*txBytes*), and bandwidth (*b*). All values correspond to the time when the packet is scheduled for transmission. At the receiver, the received packet P INT 1,2,...,r is read and the INT information is copied to the acknowledgment *ACK* packet A INT 1,2,...,r. The sender then receives an *ACK* with an INT header and metadata inserted by all the switches along the path from sender to receiver and back to sender A INT 1,2,...,r INT ..., n. Here, the INT header and meta-data pushed by switches along the path serve as feedback and as an input to the CC algorithm.

Accounting for the old window sizes: PowerTCP's control law (Eq. 7) uses the past window size in addition to the current window size to compute the new window size. PowerTCP accounts for old window size by remembering current window size once per RTT.

Algorithm: Putting it all together, we now present the workflow of PowerTCP in Algorithm 1. Upon the receipt of a new acknowledgment (line 2), PowerTCP: *(i)* retrieves the old *cwnd* (line 3), *(ii)* computes the normalized power (line 19) i.e., $\frac{f(t)}{e}$ in Eq. 7, *(iii)* updates *cwnd* (line 5), *(iv)* sets the pacing rate (line 6), and *(v)* remembers the INT header metadata and updates the old *cwnd* once per RTT based on the *ack* sequence number (line 7).

Specifically, power is calculated in the function call to NormPower. First, the gradient of queue lengths is obtained from the difference in queue lengths and difference in timestamps corresponding to an egress port (line 12). Then the transmission rate of the egress port is calculated from the difference in *txBytes* and timestamps (line 13). Current is calculated by adding the queue gradient and transmission rate (line 14). Then, the sum of BDP and the queue length gives

voltage (line 16). Finally, power is calculated by multiplying current and voltage (line 17). We calculate the base power (line 18) and obtain the normalized power (line 19). The normalized power is calculated for each egress port along the path and the maximum value is smoothed and used as an input to the control law.

Finally, the congestion window is updated in the function call to UPDATEWINDOW (line 26) where γ is the exponential moving average parameter and β is the additive increase parameter, both being parameters to the control law (Eq. 7)

Algorithm 1: PowerTCP

```
1  /* ack contains an INT header with
      sequence of per-hop egress port
      meta-data accessed as ack.H[i]        */
   Input  : ack and prevInt
   Output : cwnd, rate
2  procedure NEWACK(ack):
3      cwnd_old = GETCWND(ack.seq)
4      normPower = NORMPOWER(ack)
5      UPDATEWINDOW(normPower, cwnd_old)
6      rate = cwnd/τ
7      prevInt = ack.H; UPDATEOLD(cwnd, ack.seq)
8  function NORMPOWER(ack):
9      Γ_norm = 0
10     for each egress port i on the path do
11         dt = ack.H[i].ts − prevInt[i].ts
12         q̇ = (ack.H[i].qlen − prevInt[i].qlen)/dt      ▷ dq/dt
13         μ = (ack.H[i].txBytes − prevInt[i].txBytes)/dt   ▷ txRate
14         λ = q̇ + μ                                    ▷ λ : Current
15         BDP = ack.H[i].b × τ
16         ν = ack.H[i].qlen + BDP                      ▷ ν : Voltage
17         Γ' = λ × ν                                   ▷ Γ' : Power
18         e = (ack.H[i].b)^2 × τ
19         Γ'_norm = Γ'/e                   ▷ Γ'_norm :Normalized power
20         if Γ' > Γ_norm then
21             Γ_norm = Γ'; Δt = dt
22         end if
23     end for
24     Γ_smooth = (Γ_smooth·(τ − Δt) + Γ_norm·Δt)/τ      ▷ Smoothing
25     return Γ_smooth
26 function UPDATEWINDOW(power, ack):
27     cwnd = γ × (cwnd_old/normPower + β) + (1 − γ) × cwnd
28                                          ▷ γ: EWMA parameter
29                                          ▷ β: Additive Increase
30     return cwnd
```

Parameters: PowerTCP has only two parameters, that is the EWMA parameter γ and the additive increase parameter β. γ dictates the balance in reaction time and sensitivity to noise. We recommend $\gamma = 0.9$ based on our parameter sweep over wide range of scenarios including traffic patterns that induce rapid fluctuations in the bottleneck queue lengths. Reflecting the intuition for additive increase in prior work [36], we set $\beta = \frac{HostBw \times \tau}{N}$ where N is the expected number of flows sharing host NIC, $HostBw$ is the NIC bandwidth at the host and τ is the base-RTT. This is to avoid queuing at the local interface or, in other words, to avoid making the host NIC a bottleneck, assuming a maximum of N flows share the host NIC bandwidth. Finally, all flows transmit at line rate in the first RTT and use $cwnd_{init} = HostBw \times \tau$. By transmitting at line rate, a new flow is able to discover the bottleneck link state and reduce its *cwnd* accordingly without getting throttled due to the presence of existing flows.

3.4 Properties of PowerTCP

PowerTCP comes with strong theoretical guarantees. We show that PowerTCP's control law achieves asymptotic stability with a unique equilibrium point that satisfies our desired equilibrium state properties (Eq. 1). PowerTCP also guarantees rapid convergence to equilibrium and achieves fairness at the same time. In the following we outline PowerTCP's properties and defer the proofs to Appendix A.

Theorem 1 (Stability)**.** *PowerTCP's control law is Lyapunov-stable as well as asymptotically stable with a unique equilibrium point.*

Theorem 2 (Convergence)**.** *After a perturbation, PowerTCP's control law exponentially converges to equilibrium with a time constant* $\frac{\delta t}{\gamma}$ *where* δt *is the window update interval.*

Theorem 3 (Fairness)**.** PowerTCP *is* β_i *weighted proportionally fair, where* β_i *is the additive increase used by a flow i.*

Theorem 1 and Theorem 2 state the key properties of PowerTCP. First, the convergence with time constant of $\frac{\delta t}{\gamma}$ shows the fast reaction to perturbations. Second, the system being asymptotically stable at low queue lengths satisfies our stringent equilibrium property discussed in §2. Indeed, **power** and Property 1 play a key role in the proof of Theorem 1 and Theorem 2 (Appendix A) revealing its importance in congestion control. In Figure 3c, we see the trajectories of PowerTCP from different initial states to a unique equilibrium without violating throughput and latency requirements, showing the accurate control enabled by power-based congestion control.

3.5 θ-PowerTCP: Standalone Version

PowerTCP's control law requires in-network queue length information which can be obtained by using techniques such as INT. In order to widen its applicability, PowerTCP can still be deployed in datacenters with legacy, non-programmable switches through accurate RTT measurement capabilities at the end-host. In this case, we rearrange term $\frac{e}{f}$ in Eq. 7 as follows,

$$\frac{e}{f} = \frac{b^2 \cdot \tau}{\Gamma} = \frac{b^2 \cdot \tau}{(\dot{q} + b) \cdot (q + b \cdot \tau)} = \frac{\tau}{(\frac{\dot{q}}{b} + 1) \cdot (\frac{q}{b} + \tau)}$$

finally, using the fact that $\frac{q}{b}+\tau=\theta$ (RTT) and $\frac{\dot{q}}{b}=\dot{\theta}$ (RTT-gradient), we reduce $\frac{e}{f}$ to,

$$\frac{e}{f}=\frac{\tau}{(\dot{\theta}+1)\cdot(\theta)} \tag{8}$$

where $\dot{\theta}$ is the RTT-gradient and θ is RTT. Using Eq. 8 in Eq. 7 allows for deployment even when INT is not supported by switches in the datacenter. Due to space constraints we moved the algorithm to Appendix D, presenting θ-PowerTCP in Algorithm 2. This algorithm demonstrates how PowerTCP's control law can be mimicked by using a delay signal without the need for switch support. However, as we will show later in our evaluation, there are drawbacks in using RTT instead of queue lengths. First, notice how queue lengths are changed to RTT, where we assume bottleneck $txRate$ (μ) as bandwidth (b). The implication is that, when using $txRate$ which is essentially obtained from INT, the control law knows the exact transmission rate and rapidly fills the available bandwidth. But, when using RTT, the control law assumes the bottleneck is at maximum transmission rate and does not react by multiplicative increase and rather relies on slow additive increase to fill the available bandwidth. Secondly, in multi-bottleneck scenarios, the control law precisely reacts to the most bottlenecked link when using INT but reacts to the sum of queuing delays when using RTT. Nevertheless, under congestion, both PowerTCP and θ-PowerTCP have the same properties in a single-bottleneck scenario.

3.6 Deploying PowerTCP

Modern programmable switches are able to export user-defined header fields and device metrics [18, 32]. These metrics can be embedded into data packets, a mechanism commonly referred to as in-band network telemetry (INT). PowerTCP leverages INT to obtain fine-grained, per-packet feedback about queue occupancies, traffic counters, and link configurations within the network. For deployment with legacy networking equipment, we have proposed θ-PowerTCP which only requires accurate timestamps to measure the RTT.

We imagine PowerTCP and θ-PowerTCP to be deployed on low-latency kernel-bypass stacks such as SNAP [11] or using NIC offload. Yet, in this work, instead of implementing our algorithms for these platforms, we show how PowerTCP and θ-PowerTCP can readily be deployed by merely changing the control logic of existing congestion control algorithms. In particular, we compare our work to HPCC [36] which is based on INT feedback and SWIFT [34] which is based on delay feedback.

PowerTCP requires the same switch support and header format as HPCC, as well as packet pacing support from the NIC. Additionally, it does not maintain additional state compared to HPCC but requires one extra parameter γ, the moving average parameter for window updates. Similar to SWIFT and TIMELY, θ-PowerTCP requires accurate packet timestamps from the NIC but it does not require any switch support. The simpler logic of θ-PowerTCP (compared to PowerTCP) only reacts once per RTT and reduces the number of congestion control function calls.

The core contribution of this paper is the design of a novel control law and we do not explore implementation challenges further at this point since PowerTCP does not add additional complexity compared to existing algorithms. Still, to confirm the practical feasibility of our approach, we implemented PowerTCP as a Linux kernel congestion control module. We also implemented the INT component as a proof of concept for the Intel Tofino switch ASIC [18].

The switch implementation is written in P4 and uses a direct counter associated with the egress port to maintain the so far transmitted bytes and appends this metric together with the current queue occupancy upon dequeue from the traffic manager to each segment. We leverage a custom TCP option type to encode this data and append 64 bit per-hop headers to a 32 bit base header. The implementation uses less than one out of 12 stages of the Tofino's ingress pipeline (where the headers are prepared and appended) and less than one out of 12 stages in the egress pipeline (where the measurements are taken and inserted). The processing logic runs at line rate of 3.2 Tbit per second.

4 Evaluation

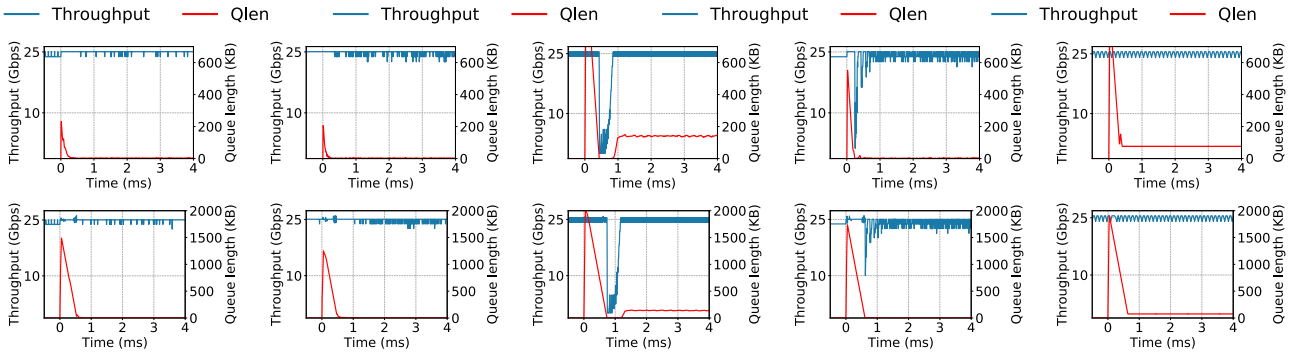
We evaluate the performance of PowerTCP and θ-PowerTCP and compare against existing CC algorithms. Our evaluation aims at answering four main questions.

(Q1) How well does PowerTCP react to congestion?
We find that PowerTCP outperforms the state-of-the-art congestion control algorithms, reducing tail buffer occupancy and consequently tail latency under congestion by 30% when compared to HPCC and at least by 60% compared to TIMELY and DCQCN.

(Q2) Does PowerTCP introduce a tradeoff between throughput and latency?
Our evaluation shows that PowerTCP does not trade throughput for latency and that PowerTCP rapidly converges to near-zero queue lengths without losing throughput.

(Q3) How much can we benefit under realistic workloads?
We show that PowerTCP improves 99th-percentile flow completion times for short flows ($< 10KB$) by 33% compared to HPCC, by 99% compared to HOMA and by 74% compared to TIMELY and DCQCN even at moderate network loads. At the same time, we find that PowerTCP does not penalize long flows ($> 1MB$). In fact, we find that θ-PowerTCP performs equally well for short flows compared to PowerTCP but performs similarly to TIMELY for medium and long flows.

(Q4) How does PowerTCP perform under high load and bursty traffic patterns?
Our evaluation shows that the benefits of PowerTCP are further enhanced under high loads and that PowerTCP remains stable even under bursty traffic.



(a) PowerTCP (b) θ-PowerTCP (c) TIMELY (d) HPCC (e) HOMA

Figure 4: State-of-the-art congestion control algorithms vs PowerTCP in response to an incast. For each algorithm, we show the corresponding reaction to 10 : 1 incast in the top row and to 255 : 1 incast in the bottom row.

4.1 Setup

Our evaluation is based on network simulator NS3 [4].

Topology: We consider a datacenter network based on a Fat-Tree topology [5] with 2 core switches and 256 servers organized into four pods. Each pod consists of two ToR switches and two aggregation switches. The capacity of all the switch-to-switch links are 100Gbps and server-to-switch links are all 25Gbps leading to 4 : 1 oversubscription similar to prior work [49]. The links connecting to core switches have a propagation delay of $5\mu s$ and all the remaining links have a propagation delay of $1\mu s$. We set up a shared memory architecture on all the switches and enable the Dynamic Thresholds algorithm [17] for buffer management across all the ports, commonly enabled in datacenter switches [1,2]. Finally we set the buffer sizes in our topology proportional to the bandwidth-buffer ratio of Intel Tofino switches [18].

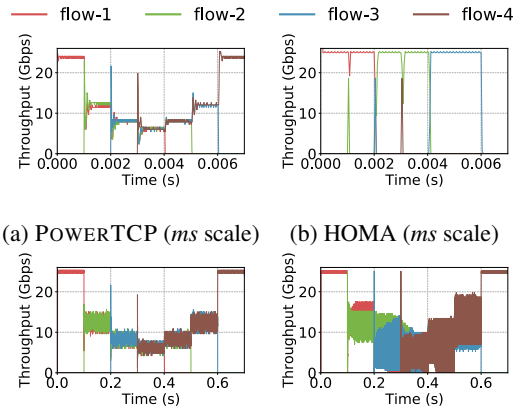
Traffic mix: We generate traffic using the web search [7] flow size distribution to evaluate our algorithm using realistic workloads. We evaluate an average load (on the ToR uplinks) in the range of $20\% - 95\%$. We also use a synthetic workload similar to prior work [6] to generate incast traffic. Specifically, the synthetic workload represents a distributed file system where each server requests a file from a set of servers chosen uniformly at random from a different rack. All the servers which receive the request respond at the same time by transmitting the requested part of the file. As a result, each file request creates an incast scenario. We evaluate across different request rates and request sizes.

Comparisons and metrics: We evaluate PowerTCP with and without switch support and compare to HPCC [36], DCQCN [56], and TIMELY [41] representing sender-based control law approaches similar to PowerTCP and HOMA [42] representing receiver-driver transport. We report flow completion times and switch buffer occupancy metrics.

Configuration: We set $\gamma = 0.9$ for PowerTCP and θ-PowerTCP. Both HPCC and PowerTCP are configured with base-RTT (τ) set to the maximum RTT in our topology and *HostBw* is set to the server NIC bandwidth. The product of base-RTT and *HostBw* is configured as RTTBytes for HOMA and the over-commitment level is set to 1 where HOMA performed best across different overcommitment levels in our setup. We report our results for all overcommitment levels (1-6) in Appendix C. We set the parameters for DCQCN following the suggestion in [36] which is based on experience and TIMELY parameters are set according to [41].

4.2 Results

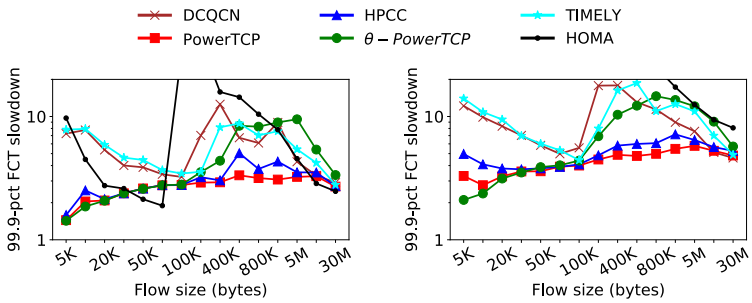
PowerTCP reacts rapidly yet accurately to congestion: We evaluate PowerTCP's reaction to congestion in two scenarios: *(i)* 10 : 1 small-scale incast and *(ii)* 255 : 1 large-scale incast. Figure 4 shows the aggregate throughput and the buffer occupancy at the bottleneck link for PowerTCP, TIMELY, HPCC and HOMA. First, at time $t = 0$, we launch ten flows simultaneously towards the receiver of a long flow leading to a **10:1** incast. We show in Figure 4a and Figure 4b that PowerTCP quickly mitigates the incast and reaches near zero queue lengths without losing throughput. In Figure 4d we see that HPCC indeed reacts quickly to get back to near-zero queue lengths. On one hand, however, HPCC does not react enough during the congestion onset and reaches higher buffer occupancy $\approx$ 2x compared to PowerTCP and on the other hand loses throughput after mitigating the incast as opposed to PowerTCP's stable throughput. TIMELY as shown in Figure 4c does not control the queue-lengths either and loses throughput after reacting to the incast. While HOMA sustains throughput, we observe from Figure 4e that HOMA does not accurately control bottleneck queue-lengths. Second, at time $t = 0$, in addition to the 10 : 1 incast, the 256^{th} server sends a query request (§4.1) to all the other 255 servers which then respond at the same time, creating a **255:1** incast. From Figure 4a and Figure 4b (bottom row), we observe similar benefits from both PowerTCP and θ-PowerTCP even at large-scale incast: both react quickly and converge to near-zero queue-lengths without losing throughput. In contrast, from Figure 4c and Figure 4d we see that TIMELY and HPCC lose throughput immediately after re-



(a) PowerTCP (*ms* scale) (b) HOMA (*ms* scale)

(c) θ-PowerTCP (d) TIMELY

Figure 5: Fairness and stability



(a) 20% load (b) 60% load

Figure 6: 99.9 percentile flow completion times with websearch workload *(a)* even at low network load, PowerTCP outperforms existing algorithms and *(b)* as the load increases the benefits of PowerTCP are enhanced. However, only short flows benefit from θ-PowerTCP.

acting to the increased queue length. From Figure 4e we observe that HOMA reaches approximately 500KB higher queue-length compared to PowerTCP and cannot converge to near-zero queue-lengths quickly.

PowerTCP is stable and achieves fairness: PowerTCP not only reacts rapidly to reduce queue lengths but also features excellent stability. Figure 5 shows how bandwidth is shared by multiple flows as they arrive and leave. We see that PowerTCP stabilizes to a fair share of bandwidth quickly, both when flows arrive and leave, confirming PowerTCP's fast reaction to congestion as well as the available bandwidth.

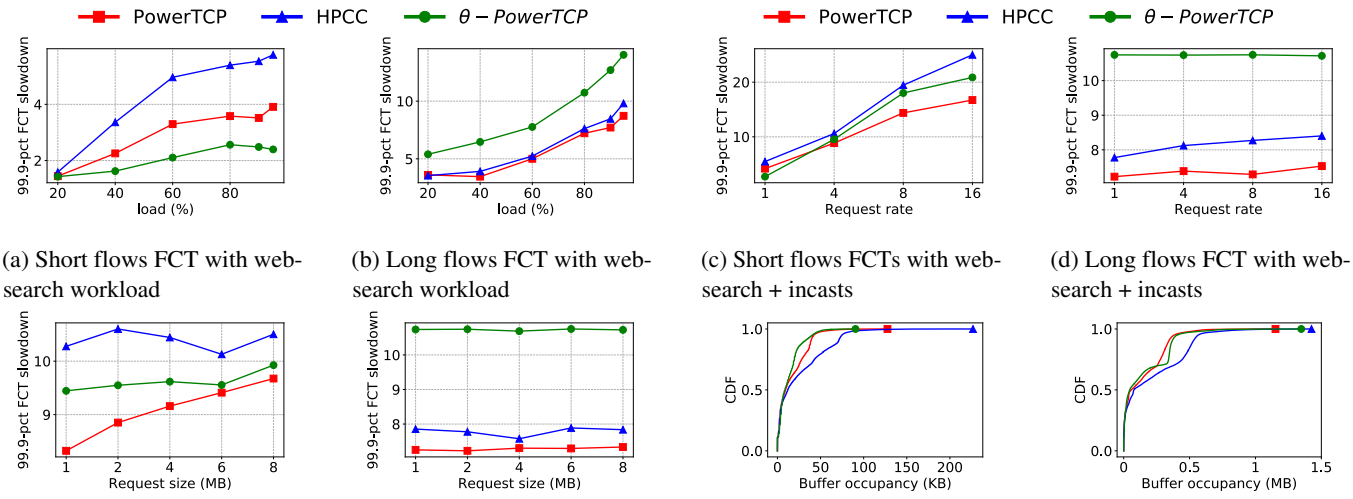
Figure 4a showing convergence and Figure 5a showing fairness and stability confirm the theoretical guarantees of PowerTCP. Hereafter, all our results are based on the setup described above, §4.1, using realistic workloads.

PowerTCP significantly improves short flows FCTs: In Figure 6 we show the 99.9-percentile flow completion times using PowerTCP and state-of-the-art datacenter congestion control algorithms. At 20% network load (Figure 6a), PowerTCP and θ-PowerTCP improve 99.9-percentile flow completion times for short flows ($< 10KB$) by 9% compared to HPCC and by 80% compared to TIMELY, DCQCN and HOMA. Even at moderate load of 60% (Figure 6b), short flows significantly benefit from PowerTCP as well as θ-PowerTCP. Specifically, PowerTCP improves 99.9 percentile flow completion times for short flows by 33% compared to HPCC, by 99% compared to HOMA and by 74% compared to TIMELY and DCQCN. θ-PowerTCP provides even greater benefits to short flows showing an improvement of 36% compared to HPCC and 82% compared to TIMELY and DCQCN. Indeed, web search workload being buffer-intensive, our results confirm the observations made in §2. TIMELY being a current-based CC, does not explicitly control queuing latency, while HPCC, a voltage-based CC, does not react as fast as PowerTCP to mitigate congestion resulting in higher flow completion times. Surprisingly, HOMA performs the worst, showing an order-of-magnitude higher FCTs for short flows at high loads as shown in Figure 6b.

We also evaluate across various loads in the range $20\% - 95\%$ and show the 99.9-percentile flow completion times for short flows in Figure 7a. In particular, we see that the benefits of PowerTCP and θ-PowerTCP are further enhanced as the network load increases. PowerTCP (and θ-PowerTCP) improve the flow completion times of short flows by 36% (and 55%) compared to HPCC. Short flows particularly benefit from PowerTCP due its accurate control of buffer occupancies close to zero. In Figure 7g we show the CDF of buffer occupancies at 80% load. PowerTCP consistently maintains lower buffer occupancy and cuts the tail buffer occupancy by 50% compared to HPCC.

Medium sized flows also benefit from PowerTCP: We find that PowerTCP not only improves short flow performance but also improves the 99.9-percentile flow completion times for medium sized flows ($100KB - 1M$). In Figure 6 we see that PowerTCP consistently achieves better flow completion times for medium sized flows. Specifically, at 20% network load (Figure 6a), PowerTCP improves 99.9-percentile flow completion times for medium flows by 33% compared to HPCC, by 76% compared to HOMA and by 62% (and 50%) compared to TIMELY (and DCQCN). In Figure 6b, we observe similar benefits even at 60% load.

We notice from Figure 6a and Figure 6b that the performance of θ-PowerTCP deteriorates sharply for medium sized flows. θ-PowerTCP uses RTT for window update calculations. While RTT can be a good congestion signal, it does not signal under-utilization as opposed to INT that explicitly notifies the exact utilization. As a result, medium flows with θ-PowerTCP experience 60% worse performance on average compared to PowerTCP and HPCC. We also observe similar performance for TIMELY that uses RTT as a congestion signal. Although delay is simple and effective for short flows performance even at the tail, our results show that delay



(a) Short flows FCT with web-search workload

(b) Long flows FCT with web-search workload

(c) Short flows FCTs with web-search + incasts

(d) Long flows FCT with web-search + incasts

(e) Short flows FCTs with web-search + incasts

(f) Long flows FCT with web-search + incasts

(g) Buffer occupancy with web-search workload at 80% load

(h) Buffer occupancy with web-search + incasts

Figure 7: A detailed comparison of PowerTCP, θ-PowerTCP and the state-of-the-art showing the benefits of PowerTCP and the trade-offs of θ-PowerTCP. Particularly PowerTCP outperforms the state-of-the-art across a range of network loads even under bursty traffic. However, θ-PowerTCP performs well for short flows but long flows cannot benefit from θ-PowerTCP.

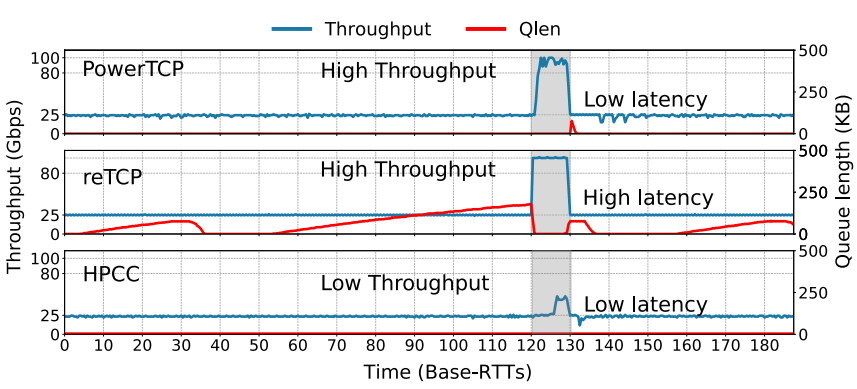
as a congestion signal is not ideal if not worse for medium sized flows.

PowerTCP does not penalize long flows: Fast reaction to available bandwidth makes PowerTCP ideal for best performance across all flow sizes. We observe from Figure 6 that PowerTCP achieves flow completion times comparable to existing algorithms, indicating that PowerTCP does not trade throughput for low latency. Further, in Figure 7b we show the 99.9-percentile flow completion times for long flows across various loads. At low load, PowerTCP performs similar to HPCC and performs 9% better compared to HPCC at 90% network loads. However, we see that θ-PowerTCP is consistently 35% worse on average across various loads compared to PowerTCP and HPCC.

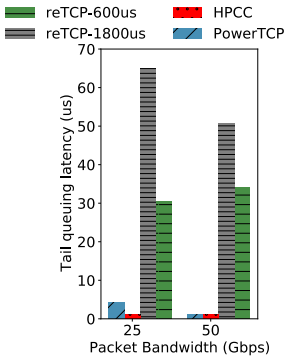
PowerTCP outperforms under bursty traffic: We generate incast-like traffic described in §4.1 in addition to the web search workload at 80% load. In Figure 7c and Figure 7d we show the 99.9-percentile flow completion times for short and long flows across different request rates for a request size of 2*MB*. Note that by varying request rates, we are essentially varying the frequency of incasts. We observe that even under bursty traffic, PowerTCP improves 99.9-percentile flow completion times on average for short flows by 24% and for long flows by 10% compared to HPCC. Further PowerTCP outperforms at high request rates showing 33% improvement over HPCC for short flows. On the other hand, θ-PowerTCP improves flows completion times for short flows but performs worse across all request rates compared to HPCC.

We further vary the request size at a request rate of four per second. Note that by varying the request size, we also vary the duration of congestion. In Figure 7e and Figure 7f, we show the 99.9-percentile flow completion times for short and long flows. Specifically, in Figure 7e we observe that flow completion times with PowerTCP gradually increase with request size. PowerTCP, compared to HPCC, improves flow completion times of short flows by 20% at 1*MB* request size and improves by 7% at 8*MB* request size. At the same time, PowerTCP does not sacrifice long flows performance under bursty traffic. PowerTCP improves flow completion times for long flows by 5% on average compared to HPCC. θ-PowerTCP's performance similar to previous experiments is on average 30% worse for long flows but 9% better for short flows compared to HPCC. We show the CDF of buffer occupancies under bursty traffic with 2*MB* request size and 16 per second request rate. Both PowerTCP and θ-PowerTCP reduce the 99 percentile buffer by 31% compared to HPCC.

We note that HOMA's performance in our evaluation is not in line with the results presented in [42]. Recent work [26] reports similar performance issues with HOMA. We suspect two possible reasons: *(i)* HOMA's accuracy in controlling congestion is specifically limited in our network setup with an oversubscribed Fat-Tree topology where congestion at the ToR uplinks is a possibility which cannot be controlled by a receiver-driven approach such as HOMA. *(ii)* As pointed out by [26], HOMA's original evaluation considered practically infinite buffers at the switches whereas switches in our setup are limited in buffer and use Dynamic Thresholds to share buffer. Further, even at 20% load, asymmetric RTTs in a Fat-Tree topology (consequently RTTBytes) across ToR pairs contributes to HOMA's inaccuracy in controlling congestion.



(a) PowerTCP reacts rapidly to the available bandwidth achieving good circuit utilization.



(b) PowerTCP significantly reduces the tail latency

Figure 8: The benefits of PowerTCP in reconfigurable datacenter networks showing its ability to achieve good circuit utilization while significantly reducing the tail latency compared to reTCP.

5 Case Study: Reconfigurable DCNs

Given PowerTCP's rapid reaction to congestion and available bandwidth, we believe that PowerTCP is well suited for emerging reconfigurable datacenter networks (RDCN) [44]. We now examine PowerTCP's applicability in this context through a case study. Congestion control in RDCNs is especially challenging as the available bandwidth rapidly fluctuates due to changing circuits. In this section, we evaluate the performance of PowerTCP and compare against the state-of-the-art reTCP [43] and HPCC using packet-level simulations in NS3. We implement both PowerTCP and HPCC in the transport layer and limit their window updates to once per RTT for a fair comparison with reTCP. PowerTCP and HPCC flows initialize the TCP header with the unused option number 36. Switches are configured to append INT metadata to TCP options. It should be noted that TCP options are limited to 40 bytes. As a result, our implementation can only support at most four hops round-trip path length.

We evaluate in a topology with 25 ToR switches with 10 servers each and a single optical circuit switch connected to all the ToR switches. ToR switches are also connected to a separate packet switched network with 25Gbps links. The optical switch internally connects each input port to an output port and cycles across 24 matchings in a permutation schedule where the switch stays in a specific matching for 225μs (one day) and takes 20μs to reconfigure to the next matching (one night). In this setting, each pair of ToR switches has direct connectivity through the circuit switch once over a length of 24 matchings (one week). We use single-hop routing in the circuit network and a maximum base RTT is 24μs. Note that circuit-on time (i.e., one day) is approximately 10 RTTs. The links between servers and ToR switches are 25Gbps and circuit links are 100Gbps. We configure the ToR switches to forward packets exclusively on the circuit network when available. Switches are further equipped with per-destination virtual output queues (VOQs). Our setup is in line with prior work [43]. We set reTCP's prebuffering to 1800μs based on the suggestions in [43] and set to 600μs based on our parameter sweep for the minimum required prebuffering in our topology. We compare against both versions.

In Figure 8a, we show the time series of throughput and VOQ length for a pair of ToR switches. Specifically, the gray-shaded area in Figure 8a highlights the availability of high bandwidth through the circuit-switched network. On one hand, reTCP instantly fills the available bandwidth but incurs high latency due to prebuffering before the circuit is available. On the other hand, HPCC maintains low queue lengths but does not fill the available bandwidth. In contrast, PowerTCP fills the available bandwidth within one RTT and maintains near-zero queue lengths and thereby achieves both high throughput and low latency. We show the tail queuing latency incurred by reTCP, HPCC and PowerTCP in Figure 8b. We observe that PowerTCP improves the tail queuing latency at least by 5× compared to reTCP. Our case study reveals that fine-grained congestion control algorithms such as PowerTCP can alleviate the circuit utilization problem in RDCNs without trading latency for throughput.

6 Related Work

Dealing with congestion has been an active research topic for decades with a wide spectrum of approaches, including buffer management [3, 10, 17] and scheduling [9, 25, 45, 46]. In the following, we will focus on the most closely related works on end-host congestion control.

Approaches such as [7, 51, 56] (e.g., DCTCP, D^{2}TCP) rely on ECN as the congestion signal and react proportionally. Such algorithms require the bottleneck queue to grow up to a certain threshold, which results in queuing delays. ECN-based schemes remain oblivious to congestion onset and intensity. Protocols such as TIMELY [41], SWIFT [34], CDG [23], DX [35] rely on RTT measurements for window update calculations. TIMELY and CDG partly react to congestion based on delay gradients, remaining oblivious to absolute queue lengths. TIMELY, for instance, uses a threshold to fall back

to proportional reaction to delay instead of delay gradient. SWIFT, a successor of TIMELY, only reacts proportionally to delay. As a result, SWIFT cannot detect congestion onset and intensity unless the distance from target delay significantly increases. In contrast, θ-PowerTCP also being a delay-based congestion control algorithm updates the window sizes using the notion of power. As a result, θ-PowerTCP accurately detects congestion onset even at near-zero queue lengths.

XCP [30], D^3 [52], *RCP* [19] rely on explicit network feedback based on rate calculations within the network. However, the rate calculations are based on heuristics and require parameter tuning to adjust for different goals such as fairness and utilization. HPCC [36] introduces a novel use of in-band network telemetry and significantly improves the fidelity of feedback. Our work builds on the same INT capabilities to accurately measure the bottleneck link state. However, as we show analytically and empirically, HPCC's control law then adjusts rate and window size solely based on observed queue lengths and lacks control accuracy compared to PowerTCP. Our proposal PowerTCP uses the same feedback signal but uses the notion of power to update window sizes leading to significantly more fine-grained and accurate reactions.

Receiver-driven transport protocols such as NDP [22], HOMA [42], and Aeolus [26] have received much attention lately. Such approaches are conceptually different from classic transmission control at the sender. Importantly, receiver-driven transport approaches make assumptions on the uniformity in datacenter topologies and oversubscription [22]. PowerTCP is a sender-based classic CC approach that uses our novel notion of power and achieves fine-grained control over queuing delays without sacrificing throughput.

7 Conclusion

We presented PowerTCP, a novel fine-grained congestion control algorithm. By reacting to both the current state of the network as well as its trend (i.e., power), PowerTCP improves throughput, reduces latency, and keeps queues within the network short. We proved that PowerTCP has a set of desirable properties, such as fast convergence and stability allowing it to significantly improve flow completion times compared to the state-of-the-art. Its fast reaction makes PowerTCP attractive for many dynamic network environments including emerging reconfigurable datacenters which served us as a case study in this paper. In our future work, we plan to explore more such use cases.

Acknowledgments

We would like to thank our shepherd, Michael Schapira, as well as the anonymous NSDI reviewers for their useful feedback. This work is part of a project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme, consolidator project Self-Adjusting Networks (AdjustNet), grant agreement No. 864228, Horizon 2020, 2020-2025.

References

[1] Broadcom. 12.8 tb/s strataxgs tomahawk 3 ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56980-series.

[2] Broadcom. 2020. 25.6 tb/s strataxgs tomahawk 4 ethernet switch series. https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56990-series.

[3] Cisco nexus 9000 series switches. https://www.cisco.com/c/en/us/products/collateral/switches/nexus-9000-series-switches/white-paper-c11-738488.html.

[4] Ns3 network simulator. https://www.nsnam.org/.

[5] Mohammad Al-Fares, Alexander Loukissas, and Amin Vahdat. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference*, page 63–74, 2008.

[6] Mohammad Alizadeh and Tom Edsall. On the data path performance of leaf-spine datacenter fabrics. In *2013 IEEE 21st annual symposium on high-performance interconnects*, pages 71--74. IEEE, 2013.

[7] Mohammad Alizadeh, Albert Greenberg, David A Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 Conference*, pages 63--74, 2010.

[8] Mohammad Alizadeh, Adel Javanmard, and Balaji Prabhakar. Analysis of dctcp: stability, convergence, and fairness. *ACM SIGMETRICS Performance Evaluation Review*, 39(1):73--84, 2011.

[9] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference*, page 435–446, 2013.

[10] Maria Apostolaki, Laurent Vanbever, and Manya Ghobadi. Fab: Toward flow-aware buffer sharing on programmable switches. In *Proceedings of the 2019 Workshop on Buffer Sizing*, pages 1--6, 2019.

[11] Mina Tahmasbi Arashloo, Yaron Koral, Michael Greenberg, Jennifer Rexford, and David Walker. Snap: Stateful network-wide abstractions for packet processing. In *Proceedings of the ACM SIGCOMM 2016 Conference*, page 29–43, 2016.

[12] Chen Avin, Manya Ghobadi, Chen Griner, and Stefan Schmid. On the complexity of traffic traces and implications. In *Proc. ACM SIGMETRICS*, 2020.

[13] Chen Avin and Stefan Schmid. Renets: Statically-optimal demand-aware networks. In *Proc. SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS)*, 2021.

[14] Hitesh Ballani, Paolo Costa, Raphael Behrendt, Daniel Cletheroe, Istvan Haller, Krzysztof Jozwik, Fotini Karinou, Sophie Lange, Kai Shi, Benn Thomsen, and Hugh Williams. Sirius: A flat datacenter network with nanosecond optical switching. In *Proceedings of the ACM SIGCOMM 2020 Conference*, page 782–797, 2020.

[15] Lawrence S Brakmo, Sean W O'Malley, and Larry L Peterson. Tcp vegas: New techniques for congestion detection and avoidance. In *Proceedings of the conference on Communications architectures, protocols and applications*, pages 24--35, 1994.

[16] Yanpei Chen, Rean Griffith, Junda Liu, Randy H Katz, and Anthony D Joseph. Understanding tcp incast throughput collapse in datacenter networks. In *Proceedings of the 1st ACM workshop on Research on enterprise networking*, pages 73--82, 2009.

[17] Abhijit K Choudhury and Ellen L Hahne. Dynamic queue length thresholds for shared-memory packet switches. *IEEE/ACM Transactions On Networking*, 6(2):130--140, 1998.

[18] Intel Corporation. Intel Tofino, 2020. Retrieved Dec. 29, 2020 from https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-series/tofino.html.

[19] Nandita Dukkipati and Nick McKeown. Why flow-completion time is the right metric for congestion control. *ACM SIGCOMM Computer Communication Review*, 36(1):59--62, 2006.

[20] Monia Ghobadi, Ratul Mahajan, Amar Phanishayee, Nikhil Devanur, Janardhan Kulkarni, Gireeja Ranade, Pierre-Alexandre Blanche, Houman Rastegarfar, Madeleine Glick, and Daniel Kilper. Projector: Agile reconfigurable data center interconnect. In *Proceedings of the ACM SIGCOMM 2016 Conference*, pages 216--229, 2016.

[21] Sangtae Ha, Injong Rhee, and Lisong Xu. Cubic: a new tcp-friendly high-speed tcp variant. *ACM SIGOPS operating systems review*, 42(5):64--74, 2008.

[22] Mark Handley, Costin Raiciu, Alexandru Agache, Andrei Voinescu, Andrew W. Moore, Gianni Antichi, and Marcin Wójcik. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the ACM SIGCOMM 2017 Conference*, page 29–42, 2017.

[23] David A Hayes and Grenville Armitage. Revisiting tcp congestion control using delay gradients. In *International Conference on Research in Networking*, pages 328--341. Springer, 2011.

[24] Christopher V Hollot, Vishal Misra, Don Towsley, and Wei-Bo Gong. A control theoretic analysis of red. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No. 01CH37213)*, volume 3, pages 1510--1519. IEEE, 2001.

[25] Chi-Yao Hong, Matthew Caesar, and P Brighten Godfrey. Finishing flows quickly with preemptive scheduling. *ACM SIGCOMM Computer Communication Review*, 42(4):127--138, 2012.

[26] Shuihai Hu, Wei Bai, Gaoxiong Zeng, Zilong Wang, Baochen Qiao, Kai Chen, Kun Tan, and Yi Wang. Aeolus: A building block for proactive transport in datacenters. In *Proceedings of the ACM SIGCOMM 2020 Conference*, page 422–434, 2020.

[27] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, 1988.

[28] Cheng Jin, David X Wei, and Steven H Low. Fast tcp: motivation, architecture, algorithms, performance. In *IEEE INFOCOM 2004*, volume 4, pages 2490--2501. IEEE, 2004.

[29] Srikanth Kandula, Sudipta Sengupta, Albert Greenberg, Parveen Patel, and Ronnie Chaiken. The nature of data center traffic: measurements & analysis. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement*, pages 202--208, 2009.

[30] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 89--102, 2002.

[31] Srinivasan Keshav. *Mathematical foundations of computer networking*. Addison-Wesley, 2012.

[32] Changhoon Kim, Anirudh Sivaraman, Naga Katta, Antonin Bas, Advait Dixit, and Lawrence J Wobker. In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM '15 Demos*, 2015.

[33] Janardhan Kulkarni, Stefan Schmid, and Pawel Schmidt. Scheduling opportunistic links in two-tiered reconfigurable datacenters. In *33rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, 2021.

[34] Gautam Kumar, Nandita Dukkipati, Keon Jang, Hassan M. G. Wassel, Xian Wu, Behnam Montazeri, Yaogong Wang, Kevin Springborn, Christopher Alfeld, Michael Ryan, David Wetherall, and Amin Vahdat. Swift: Delay is simple and effective for congestion control in the datacenter. In *Proceedings of the ACM SIGCOMM 2020 Conference*, page 514–528, 2020.

[35] Changhyun Lee, Chunjong Park, Keon Jang, Sue Moon, and Dongsu Han. Accurate latency-based congestion feedback for datacenters. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 403--415, Santa Clara, CA, July 2015. USENIX Association.

[36] Yuliang Li, Rui Miao, Hongqiang Harry Liu, Yan Zhuang, Fei Feng, Lingbo Tang, Zheng Cao, Ming Zhang, Frank Kelly, Mohammad Alizadeh, et al. Hpcc: High precision congestion control. In *Proceedings of the ACM SIGCOMM 2019 Conference*, pages 44--58, 2019.

[37] S.H. Low, F. Paganini, and J.C. Doyle. Internet congestion control. *IEEE Control Systems Magazine*, 22(1):28--43, 2002.

[38] William M Mellette, Rajdeep Das, Yibo Guo, Rob McGuinness, Alex C Snoeren, and George Porter. Expanding across time to deliver bandwidth efficiency and low latency. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 1--18, 2020.

[39] William M Mellette, Rob McGuinness, Arjun Roy, Alex Forencich, George Papen, Alex C Snoeren, and George Porter. Rotornet: A scalable, low-complexity, optical datacenter network. In *Proceedings of the ACM SIGCOMM 2017 Conference*, pages 267--280, 2017.

[40] Vishal Misra, Wei-Bo Gong, and Don Towsley. Fluid-based analysis of a network of aqm routers supporting tcp flows with an application to red. In *Proceedings of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 151--160, 2000.

[41] Radhika Mittal, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, David Wetherall, and David Zats. Timely: Rtt-based congestion control for the datacenter. In *Proceedings of the ACM SIGCOMM 2015 Conference*, page 537–550, 2015.

[42] Behnam Montazeri, Yilong Li, Mohammad Alizadeh, and John Ousterhout. Homa: A receiver-driven low-latency transport protocol using network priorities. In *Proceedings of the ACM SIGCOMM 2018 Conference*, page 221–235, 2018.

[43] Matthew K Mukerjee, Christopher Canel, Weiyang Wang, Daehyeok Kim, Srinivasan Seshan, and Alex C Snoeren. Adapting TCP for reconfigurable datacenter networks. In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 651--666, 2020.

[44] Matthew Nance Hall, Klaus-Tycho Foerster, Stefan Schmid, and Ramakrishnan Durairajan. A survey of reconfigurable optical networks. *Optical Switching and Networking*, 41:100621, 2021.

[45] Jonathan Perry, Hari Balakrishnan, and Devavrat Shah. Flowtune: Flowlet control for datacenter networks. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*, pages 421--435, Boston, MA, March 2017. USENIX Association.

[46] Jonathan Perry, Amy Ousterhout, Hari Balakrishnan, Devavrat Shah, and Hans Fugal. Fastpass: A centralized" zero-queue" datacenter network. In *Proceedings of the ACM SIGCOMM 2014 conference*, pages 307--318, 2014.

[47] Amar Phanishayee, Elie Krevat, Vijay Vasudevan, David G Andersen, Gregory R Ganger, Garth A Gibson, and Srinivasan Seshan. Measurement and analysis of tcp throughput collapse in cluster-based storage systems. In *FAST*, volume 8, pages 1--14, 2008.

[48] Arjun Roy, Hongyi Zeng, Jasmeet Bagga, George Porter, and Alex C Snoeren. Inside the social network's (datacenter) network. In *Proceedings of the ACM SIGCOMM 2015 Conference*, pages 123--137, 2015.

[49] Ahmed Saeed, Varun Gupta, Prateesh Goyal, Milad Sharif, Rong Pan, Mostafa Ammar, Ellen Zegura, Keon Jang, Mohammad Alizadeh, Abdul Kabbani, and Amin Vahdat. Annulus: A dual congestion control loop for datacenter and wan traffic aggregates. In *Proceedings of the ACM SIGCOMM 2020 Conference*, page 735–749, 2020.

[50] Stefan Schmid, Chen Avin, Christian Scheideler, Michael Borokhovich, Bernhard Haeupler, and Zvi Lotker. Splaynet: Towards locally self-adjusting networks. *IEEE/ACM Transactions on Networking (ToN)*, 2016.

[51] Balajee Vamanan, Jahangir Hasan, and T.N. Vijaykumar. Deadline-aware datacenter tcp (d2tcp). In *Proceedings of the ACM SIGCOMM 2012 Conference*, page 115–126, 2012.

[52] Christo Wilson, Hitesh Ballani, Thomas Karagiannis, and Ant Rowtron. Better never than late: Meeting deadlines in datacenter networks. In *Proceedings of the ACM SIGCOMM 2011 Conference*, page 50–61, 2011.

[53] Jackson Woodruff, Andrew W Moore, and Noa Zilberman. Measuring burstiness in data center applications. In *Proceedings of the 2019 Workshop on Buffer Sizing*, 2019.

[54] Doron Zarchy, Radhika Mittal, Michael Schapira, and Scott Shenker. Axiomatizing congestion control. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):1--33, 2019.

[55] Qiao Zhang, Vincent Liu, Hongyi Zeng, and Arvind Krishnamurthy. High-resolution measurement of data center microbursts. In *Proceedings of the 2017 Internet Measurement Conference*, pages 78--85, 2017.

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

[57] Yibo Zhu, Monia Ghobadi, Vishal Misra, and Jitendra Padhye. 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*, pages 313--327, 2016.

A Analysis

Our analysis is based on a a single bottleneck link model widely used in the literature [24, 40, 54, 57]. Specifically, we assume that all senders use the same protocol, transmit long flows sharing a common bottleneck link with bandwidth b, and have a base round trip time τ (excluding queuing delays). We denote at time t queue length as $q(t)$, aggregate window size as $w(t)$, window size of a sender i as $w_i(t)$, forward propagation delay between sender and bottleneck queue as t^f, the round-trip time as $\theta(t)$ and a base round-trip time as τ. Here $w(t) = \sum_i w_i(t)$.

We additionally use the traditional model of queue length dynamics which is independent of the control law [24, 40]

$$\dot{q}(t) = \frac{w(t-t^f)}{\theta(t)} - b \tag{9}$$

where $\theta(t)$ is given by,

$$\theta(t) = \frac{q(t)}{b} + \tau \tag{10}$$

Power at time t denoted by $\Gamma(t)$ as defined in §3.1 is expressed as,

$$\Gamma(t) = \underbrace{(q(t) + b\cdot\tau)}_{voltage} \cdot \underbrace{(\dot{q}(t) + \mu(t))}_{current} \tag{11}$$

PowerTCP's control law at a source i is given by,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t-\theta(t))\cdot e}{f(t)} + \beta \right) + (1-\gamma)\cdot w_i(t) \tag{12}$$

where e and $f(t)$ are given by,

$$e = b^2 \cdot \tau$$

$$f(t) = \Gamma(t - \theta(t) + t^f)$$

and β is the additive increase term and $\gamma \in (0, 1]$ serves as the weight given for new updates using EWMA. Both β and γ are parameters to the control law.

Using the properties of power (Property 1), the aggregate window size at time $t - \theta(t)$ can be expressed in terms of power as,

$$w(t-\theta(t)) = \frac{\Gamma(t-\theta(t)+t^f)}{b} = \frac{f(t)}{b} \tag{13}$$

Suppose an *ack* arrives at time t acknowledging a segment, time $t - \theta(t)$ corresponds to the time when the acknowledged segment was transmitted.

Theorem 1 (Stability)**.** PowerTCP*'s control law is Lyapunov-stable as well as asymptotically stable with a unique equilibrium point.*

Notation	Description
b	bottleneck bandwidth
q	bottleneck queue length
τ	base RTT
t^f	sender to bottleneck delay
θ	round trip time RTT
w_i	window size of a flow i
w	aggregate window size (of all flows)
γ	EWMA parameter
β	additive increase
e	desired equilibrium point
f	feedback
λ_i	sending rate of a flow i
λ	Current: aggregate sending rate
ν	Voltage
Γ	Power

Table 2: Key notations used in this paper. Additionally for any variable say x, $\dot{x}$ denotes its derivative with respect to time i.e., $\frac{dx}{dt}$.

Proof. First, we rewrite Eq. 12 as follows to obtain the aggregate window w,

$$\sum_i w_i(t+\delta t) = \sum_i \gamma \cdot \left(\frac{w_i(t-\theta(t)) \cdot e}{f(t)} + \beta \right) + \sum_i (1-\gamma) \cdot w_i(t)$$

let $\hat{\beta} = \sum_i \beta$

$$w(t+\delta t) = \gamma \cdot \left(\frac{w(t-\theta(t)) \cdot e}{f(t)} + \hat{\beta} \right) + (1-\gamma) \cdot w(t)$$

by rearranging the terms in the above equation we obtain,

$$w(t+\delta t) - w(t) = \gamma \cdot \left(-w(t) + \frac{w(t-\theta(t)) \cdot e}{f(t)} + \hat{\beta} \right)$$

dividing by δt on both sides in the above equation and using Euler's first-order approximation, we derive the window dynamics for PowerTCP as follows,

$$\dot{w}(t) = \gamma_r \cdot \left(-w(t) + \frac{w(t-\theta(t)) \cdot e}{f(t)} + \hat{\beta} \right) \tag{14}$$

where $\gamma_r = \frac{\gamma}{\delta t}$. Using Eq. 13 and substituting $e = b^2 \cdot \tau$, Eq. 14 reduces to,

$$\dot{w}(t) = \gamma_r \cdot \left(-w(t) + b \cdot \tau + \hat{\beta} \right) \tag{15}$$

In the system defined by Eq. 9 and Eq. 14, when the window and the queue length stabilize i.e., $\dot{w}(t) = 0$ and $\dot{q}(t) = 0$, it is easy to observe that there exists a unique equilibrium point $(w_e, q_e) = (b \cdot \tau + \hat{\beta}, \hat{\beta})$. We now apply a change of variable from t to $t - t^f$ in Eq. 15 and linearize Eq. 15 and Eq. 9 around (w_e, q_e),

$$\delta\dot{w}(t-t^f) = -\gamma_r \cdot \delta w(t-t^f) \tag{16}$$

$$\delta\dot{q}(t) = -\frac{\delta q(t)}{\tau} + \frac{\delta w(t-t^f)}{\tau} \tag{17}$$

We now convert the above differential equations to matrix form,

$$\begin{bmatrix} \delta\dot{q}(t) \\ \delta\dot{w}(t) \end{bmatrix} = \begin{bmatrix} -\frac{1}{\tau} & \frac{1}{\tau} \\ 0 & -\gamma_r \end{bmatrix} \times \begin{bmatrix} \delta q(t) \\ \delta w(t) \end{bmatrix}$$

It is then easy to observe that the eigenvalues of the system are $-\frac{1}{\tau}$ and $-\gamma_r$. Since τ (base RTT) and $\gamma_r = \frac{\gamma}{\delta t}$ are both positive, we see that both the eigenvalues are negative. This proves that the system is both lyapunav stable and asymptotically stable. □

Theorem 2 (Convergence)**.** *After a perturbation,* PowerTCP*'s control law exponentially converges to equilibrium with a time constant* $\frac{\delta t}{\gamma}$ *where* δt *is the window update interval.*

Proof. A perturbation at time $t = 0$ causes the window to shift from $w_e = c \cdot \tau + \hat{\beta}$ to say w_{init}. We solve the differential equation in Eq. 15 and obtain the following equation,

$$w(t) = w_e + \underbrace{(w_{init} - w_e) \cdot e^{-\gamma_r \cdot t}}_{exponential\ decay} \tag{18}$$

From Eq. 18 we can see that, for any error $e = w_e - w_{init}$ caused by a perturbation, e exponentially decays with a time constant $\frac{1}{\gamma_r} = \frac{\delta t}{\gamma}$. Hence for e to decay 99.3%, it takes $\frac{5 \cdot \delta}{\gamma}$ time. □

Theorem 3 (Fairness)**.** PowerTCP *is* β_i *weighted proportionally fair, where* β_i *is the additive increase used by a flow* i*.*

Proof. Recall that PowerTCP's control law for each flow i is defined as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t-\theta(t)) \cdot e}{f(t)} + \beta_i \right) + (1-\gamma) \cdot w_i(t)$$

From the proof of Theorem 1, we know that the equilibrium point for aggregate window size and queue length is $(w_e, q_e) = (b \cdot \tau + \hat{\beta}, \hat{\beta})$. Using this equilibrium we can also obtain the equilibrium value for $f(t)$ as,

$$f_e = (\hat{\beta} + b \cdot \tau) \cdot b$$

We can then show that w_i has an equilibrium point.

$$(w_i)_e = \frac{\hat{\beta} + b \cdot \tau}{\hat{\beta}} \cdot \beta_i$$

We use the argument that window sizes and rates are synonymous especially that PowerTCP uses pacing with rate $r_i = \frac{w_i}{\tau}$. We can then easily observe that the rate allocation is approximately max-min fair if β_i are small enough but β_i proportionally fair in general. □

B Justifying the Simplified Model

We considered a simplified control law model to study existing control laws in §2. Here we justify how the simplified model approximately captures the existing control laws. Our simplified model for congestion window update at time $t+\delta t$ is defined in Eq. 19 as a function of current congestion window size, a target e, the feedback $f(t)$, an additive increase β and an exponential moving average parameter γ.

$$w_i(t+\delta t) = \underbrace{\gamma \cdot \underbrace{\left(w_i(t) \cdot \frac{e}{f(t)} + \beta\right)}_{update} + (1-\gamma) \cdot w_i(t)}_{EWMA} \quad (19)$$

where e and $f(t)$ are given by,

$$e = \begin{cases} b \cdot \tau & queue\text{-}length\ based\ CC \\ \tau & delay\text{-}based\ CC \\ 1 & RTT\text{-}gradient\ based\ CC \end{cases} \quad (20)$$

$$f(t) = \begin{cases} q(t-\theta(t)+t^f) + b \cdot \tau & queue\text{-}length\ based\ CC \\ \frac{q(t-\theta(t)+t^f)}{b} + \tau & delay\text{-}based\ CC \\ \frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1 & RTT\text{-}gradient\ based\ CC \end{cases} \quad (21)$$

We first use Euler's first order approximation and obtain the aggregate window ($\sum w$) dynamics for the simplified model,

$$\dot{w}(t) = \frac{\gamma}{\delta t} \cdot \left(w(t) \cdot \frac{e}{f(t)} - w(t) + \beta\right) \quad (22)$$

In order for the system to stabilize, we require $\dot{q}(t) = 0$ and $\dot{w}(t) = 0$. Using Eq. 9 and Eq. 22 and applying equilibrium conditions and assuming that $f(t)$ stabilizes,

$$q_e = w_e - b \cdot \tau \quad (23)$$

$$w_e = \frac{\hat{\beta}}{1 - \frac{e}{f}} \quad (24)$$

Recall that $\hat{\beta} = \sum_{\beta_i}$, the sum of additive increase terms of all flows sharing a bottleneck. To show whether there exists a unique equilibrium point, it remains to show whether Eq. 23 and Eq. 24 have a unique solution for w_e and q_e. We now show how the simplified model captures existing control laws and show the equilibrium properties.

Queue length or inflight-based control law: Substituting $e = b \cdot \tau$ and $f(t) = q(t-\theta(t)+t^f) + b \cdot \tau$, we express the simplified queue length based control law as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t) \cdot b \cdot \tau}{q(t-\theta(t)+t^f) + b \cdot \tau} + \beta\right) + (1-\gamma) \cdot w_i(t) \quad (25)$$

notice that the update is an MIMD based on inflight bytes. Eq. 25 captures control laws based on inflight bytes; for example HPCC [36].

A system defined by queue length based control law (Eq. 25 and the queue length dynamics (Eq. 9, there exists a unique equilibrium point. It can be observed that Eq. 24 for queue length based control law gives $w_e = b \cdot \tau + \hat{\beta}$ and $q_e = \hat{\beta}$.

Delay-based control law: Substituting $e = \tau$ and $f(t) = \frac{q(t-\theta(t)+t^f)}{b} + \tau$, we express the simplified delay-based control law as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t) \cdot \tau}{\frac{q(t-\theta(t)+t^f)}{b} + \tau} + \beta\right) + (1-\gamma) \cdot w_i(t) \quad (26)$$

where the window update is an MIMD based on RTT. Eq. 26 captures control laws based on RTT; for example FAST [28].

Similar to queue-length based CC, a system defined by delay-based control law (Eq. 26 and the queue length dynamics (Eq. 9, there exists a unique equilibrium point. It can be observed that Eq. 24 for delay-based control law gives $w_e = b \cdot \tau + \hat{\beta}$ and $q_e = \hat{\beta}$.

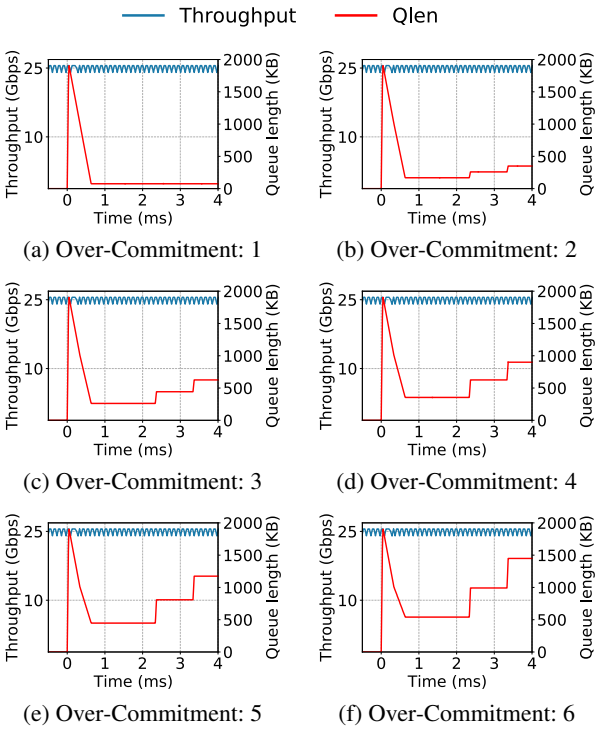
RTT-gradient based control law: Substituting $e = 1$ and $f(t) = \frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1$, we express the simplified RTT-gradient based control law as,

$$w_i(t+\delta t) = \gamma \cdot \left(\frac{w_i(t) \cdot 1}{\frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1} + \beta\right) + (1-\gamma) \cdot w_i(t) \quad (27)$$

where the window update is an MIMD based on RTT-gradient. Eq. 27 by rearranging the terms, captures control laws based on RTT-gradient such as TIMELY [41].

In contrast to queue-length and delay-based CC, RTT-gradient based CC has no unique equilibrium point since $f(t) = \frac{\dot{q}(t-\theta(t)+t^f)}{b} + 1$ stabilizes when $\dot{q} = 0$. However only $\dot{q} = 0$ leads to window dynamics Eq. 27 also to stabilize ($\dot{w} = 0$) at any queue lengths. As a result under RTT-gradient control law, Eq. 23 and Eq. 24 do not have a unique solution and consequently we can state that RTT-gradient based CC has no unique equilibrium point.

C HOMA's Overcommitment

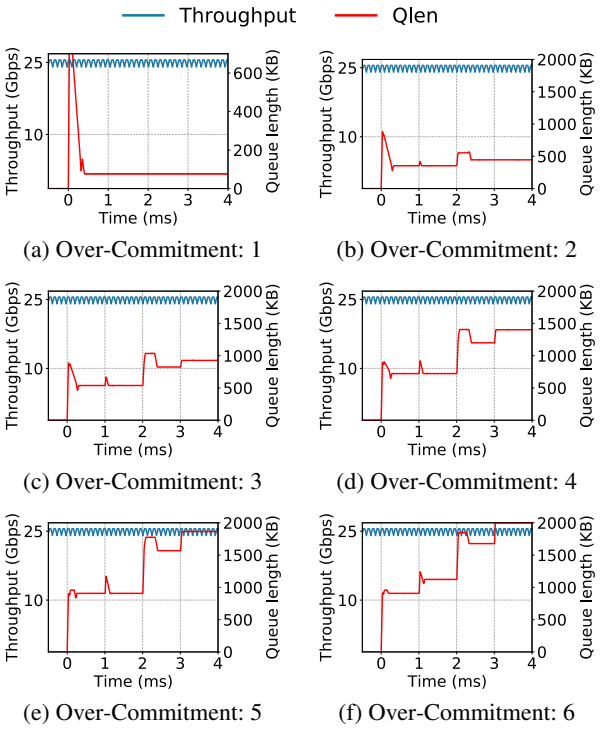


(a) Over-Commitment: 1 (b) Over-Commitment: 2

(c) Over-Commitment: 3 (d) Over-Commitment: 4

(e) Over-Commitment: 5 (f) Over-Commitment: 6

Figure 9: HOMA's reaction to 255 : 1 incast at different over-commitment levels.

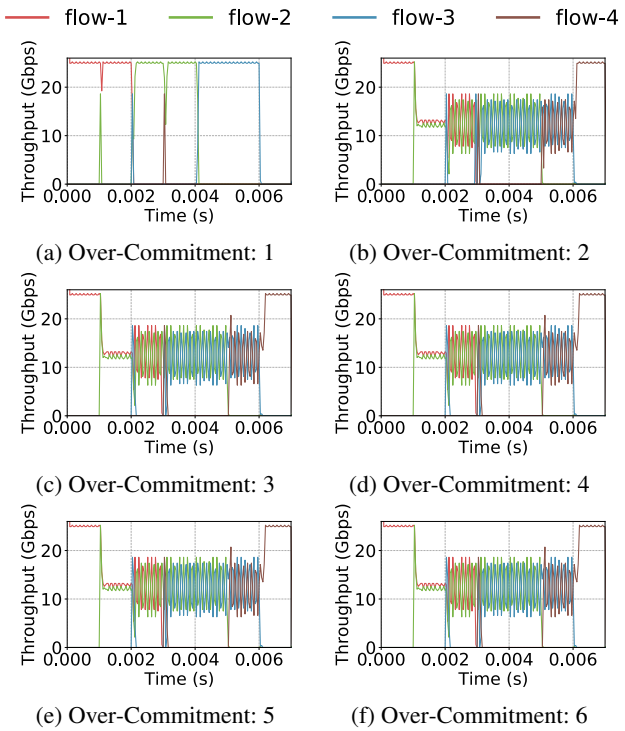


(a) Over-Commitment: 1 (b) Over-Commitment: 2

(c) Over-Commitment: 3 (d) Over-Commitment: 4

(e) Over-Commitment: 5 (f) Over-Commitment: 6

Figure 10: HOMA's reaction to 10 : 1 incast at different over-commitment levels.



(a) Over-Commitment: 1 (b) Over-Commitment: 2

(c) Over-Commitment: 3 (d) Over-Commitment: 4

(e) Over-Commitment: 5 (f) Over-Commitment: 6

Figure 11: HOMA's fairness at different over-commitment levels.

D θ-PowerTCP

We present θ-PowerTCP: standalone version of PowerTCP which does not require switch support and only requires accurate packet timestamp support at the end-host.

Algorithm 2: θ-PowerTCP (w/o switch support)

```
/* t_c is the timestamp upon ack arrival */
Input  : ack
Output : cwnd, rate
procedure NewAck(ack):
    cwnd_old = getCwnd(ack.seq)
    normPower = normPower(ack)
    updateWindow(normPower, cwnd_old)
    rate = cwnd/τ
    prevRTT = RTT
    t_c^prev = t_c
    updateOld(cwnd, ack.seq)
function normPower(ack):
    dt = t_c − t_c^prev
    θ̇ = (RTT − prevRTT)/dt                          ▷ dRTT/dt
    Γ_norm = ((θ̇ + 1) × RTT)/τ        ▷ Γ_norm :Normalized power
    Γ_smooth = (Γ_smooth · (τ − Δt) + Γ_norm · Δt)/τ
    return Γ_smooth
function updateWindow(power, ack):
    if ack.seq < lastUpdated then                   ▷ per RTT
        return cwnd
    end if
    cwnd = γ × ((cwnd_old/normPower) + β) + (1 − γ) × cwnd
                                          ▷ γ: EWMA parameter
                                          ▷ β: Additive Increase
    lastUpdated = snd_nxt
    return cwnd
```