

Delay Bounding Congestion Control Algorithms

Lecture 9
Subir Varma

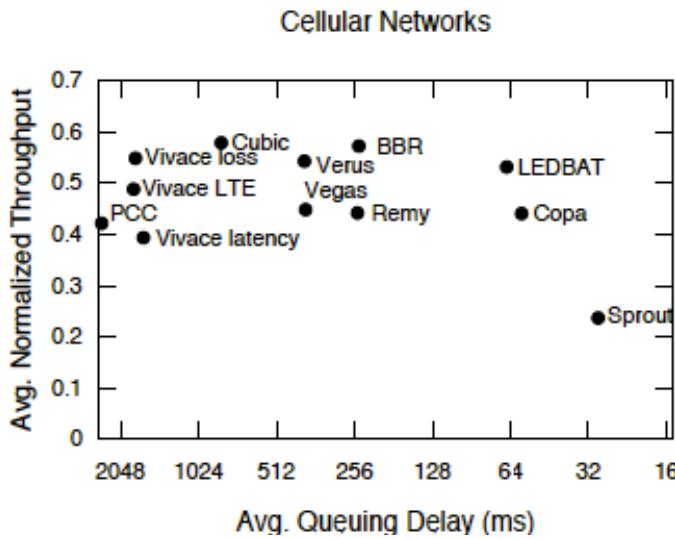
Delay Bounding Algorithms

- ▶ Delay bounding algorithms have been inspired by TCP Vegas
- ▶ TCP Vegas gets swamped by loss based algorithms (TCP Reno and its successors), and hence for a long time this class of algorithms was not very popular.
- ▶ Some popular algorithms combined delay based and loss based paradigms, such as Compound TCP and Yeah TCP.
- ▶ TCP FAST was another well known algorithm that was proposed in 2004 which tried to solve some of the issues with TCP Vegas.
- ▶ Over the past ten years, however, delay-bounding algorithms have experienced a resurgence with several proposals that overcome these issues, including Sprout, Remy (Lecture 10), BBR (Lecture 7), PCC (Lecture 10), Copa (this lecture), and Verus.

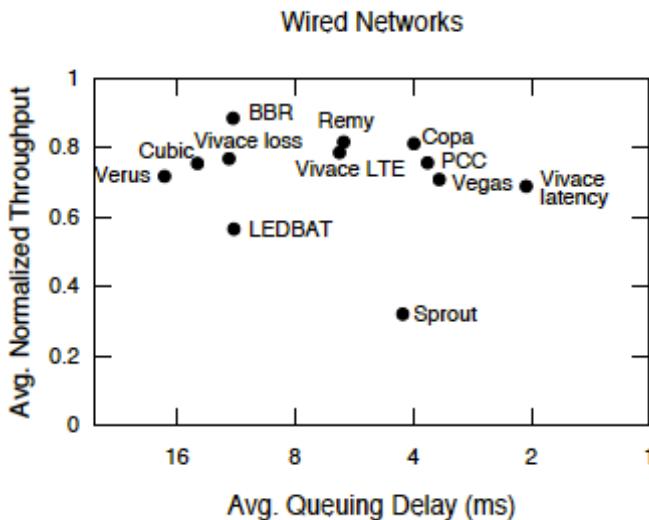
Delay Bounding Algorithms Discussed in Earlier Lectures

- ▶ Combine Loss type algorithms such as CUBIC (Lecture 6) with Explicit Congestion Notification (ECN) and trigger ECNs if the bottleneck queue size exceeds a threshold. DCTCP (Lecture 8) also belongs to this class of algorithms.
- ▶ AQM Algorithms that try to control bottleneck queue size: RED, Proportional Control, Proportional + Integral Control (all in Lecture 3), CoDel (Lecture 4).
 - All these algorithms try to bound delay by trying to keep the bottleneck queue size within bounds using router based AQM mechanisms.
- ▶ Yeah TCP (Lecture 6)
 - Yeah TCP was an improvement on Vegas: Use of fast window increase rule if the queue was below threshold, and once the the threshold was exceeded, it treated the event like a packet loss and reduced its window (and thus rate). It also has a system for competing with loss based algorithms.
- ▶ BBR (Lecture 7)
 - BBR can also be considered to be an improvement on Vegas. It tries to match transmission rate with the bottleneck bandwidth, and ignores packet losses as a measure of congestion.

So What is Left?



- We would like to optimize for throughput and delay together.
- With this criteria, the best algorithms are the ones whose performance lies in the upper right corners in these graphs.
- New congestion control schemes proposed in the last ten years are characterized by the fact that they take this joint constraint into account.
- There is also less emphasis on being compatible with Reno or CUBIC



Experimental results from running various algorithms on live testbeds using the Pantheon framework.

Issues with Delay Based Algorithms

Delay based algorithms are very good at bounding end-to-end latency, but run into several issues in real world deployments:

- ▶ How to co-exist with loss based algorithms:
 - When loss based algorithms overload the buffer, delay based algorithms back off
 - Conversely if a node is experiencing congestion losses, loss based algorithms will back off while delay based algorithms keep transmitting thus making the losses worse.
- ▶ Measuring the queueing delay requires a very accurate estimate of the minimum round trip latency T .
 - If this estimate is not accurate, it can result in an under-estimation of the delay, thus resulting in intra-protocol unfairness.
 - However getting an accurate estimate of T is not straightforward, it requires periodic draining of all queues along the path.

Issues with Delay Based Algorithms (cont)

- ▶ Early delay based algorithms such as Vegas ramped up their congestion window too slowly to be able to function in high speed networks.
 - The congestion window needs to be ramped by more than 1 packet per RTT in high bandwidth environments
- ▶ Small in-accuracies in the measurement of the queueing delay can translate into huge differences in the source rate.
 - For example if the source rate R is chosen so as to maintain a queueing size of D then

$$R(T_s - T) \approx D \text{ i.e. } R \approx \frac{D}{T_s - T}$$

Note that $(T_s - T)$ is a very small number, of the order of milliseconds, and even a small in-accuracy in its measurement can lead to huge differences in the source rate R. This number becomes smaller in higher speed networks.

TCP Vegas: A Delay based Algorithm

- ▶ TCP Vegas estimates the level of congestion in the network by calculating the difference in the expected and actual data rates, which it then uses to adjust the TCP window size. Assuming a window size of W and a minimum round trip latency of T seconds, the source computes an expected throughput R_E once per round trip delay, by

$$R_E = \frac{W}{T}$$

- ▶ The source also estimates the current throughput R by using the actual round trip time T_s (estimated using a LPF) according to

$$R = \frac{W}{T_s}$$

- ▶ The source then computes the quantity Diff given by

$$Diff = T(R_E - R) = R(T_s - T)$$

- ▶ By Little's law, $R(T_s - T)$ equals the number of packets belonging to the connection that are queued in the network and hence serves as a measure of congestion.

TCP Vegas: A Delay based Algorithm

Define two thresholds α and β such that $\alpha < \beta$. The window increase decrease rules are given by:

- When $\alpha \leq \text{Diff} \leq \beta$, then leave W unchanged.
- When $\text{Diff} > \beta$, decrease W by 1 for the next RTT. This condition implies that congestion is beginning to build up the network; hence, the sending rate should be reduced.
- When $\text{Diff} < \alpha$, increase W by 1 for the next RTT. This condition implies the actual throughput is less than the expected throughput; hence, there is some danger that the connection may not use the full network bandwidth.

Issues with TCP Vegas

- ▶ What if T is not the minimum round trip latency due to cross traffic other flows? In this case Vegas under-estimates the congestion queue length estimate $Diff$. In general estimating T requires that all flows synchronize themselves periodically so that there is no traffic on the links during the measurement process.
- ▶ Using a LPF to measure the actual latency is also problematic: It takes time for the filter to settle thus it cannot keep track of fast queue variations. Also it is the persistent delay that is important, not delays due to transient bursts.
- ▶ The presence of other loss based flows causes Vegas to back-off, thus causing inter-protocol unfairness.
- ▶ Vegas increment and decrements its window in steps of 1 packet, which is a problem in high speed links.
- ▶ There is potential for un-fairness between multiple Vegas flows sharing a link (since a Vegas flow does not back-off significantly once it reaches steady state).

TCP FAST

FAST TCP:

From Theory to Experiments ^{*}

C. Jin, D. Wei, S. H. Low

G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle
W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, S. Singh [†]

<http://netlab.caltech.edu/FAST/>

November 1, 2004

Abstract

We describe a variant of TCP, called FAST, that can sustain high throughput and utilization at multi-Gbps over large distance. We present the motivation, review the background theory, summarize key features of FAST TCP, and report our first experimental results.

Keywords: FAST TCP, large bandwidth-delay product

1 Introduction

The congestion control algorithm in the current TCP has performed remarkably well and is generally believed to have prevented severe congestion as the Internet scaled up by six orders of magnitude in size, speed, load, and connectivity in the last fifteen years. It is also well-known, however, that as bandwidth-delay product continues to grow, the current TCP implementation will eventually become a performance bottleneck.

In this paper we describe a different congestion control algorithm for TCP, called FAST [1]. FAST TCP has three key differences. First, it is an equation-based algorithm and hence eliminates packet-level oscillations. Second, it uses queuing delay as the primary measure of congestion, which can be more reliably measured by end hosts than loss probability in fast long-distance networks. Third, it has a stable flow dynamics and achieves weighted proportional fairness in equilibrium that does not penalize long flows, as the current congestion control algorithm does. Alternative approaches are described in [2, 3, 4, 5, 6]. The details of the architecture, algorithms, extensive experimental evaluations of FAST TCP, and comparison with other TCP variants can be found in [1, 7].

In this paper, we will highlight the motivation, background theory, implementation and our first major experimental results. The scientific community is singular in its urgent need for efficient high speed data transfer. We explain in Section 2 why this community has been driving the development and deployment of ultrascale networking. The design of FAST TCP builds on an emerging theory that allows us to understand the equilibrium and stability properties of large networks under end-to-end control. It provides a framework to understand issues, clarify ideas and suggest directions, leading to a more robust and better performing design. We summarize this theory in Section 3 and explain FAST

^{*} IEEE Network, to appear.

[†]G. Buhrmaster and L. Cottrell are with SLAC (Stanford Linear Accelerator Center), Stanford, CA. W. Feng is with LANL (Los Alamos National Lab). O. Martin is with CERN (European Organization for Nuclear Research), Geneva. F. Paganini is with EE Department, UCLA. All other authors are with Caltech, Pasadena, CA.

FAST TCP

- ▶ The design of FAST TCP was inspired by that of TCP Vegas. It uses end-to-end delay as a measure of congestion rather than dropped packets and can be considered to be a high speed version of Vegas.
- ▶ FAST TCP has 4 components:
 - Estimation
 - Window Control
 - Data Control
 - Burstiness Control

TCP FAST: Estimation

This component computes two pieces of information for each data packet sent:

- ▶ Multi-bit Queueing delay: This is estimated by measuring the minimum RTT for a connection (called baseRTT) and also computing an exponentially smoothed average RTT (called $T_i(t)$). Note that
 $T_i(t) = \text{baseRTT} + q_i(t)$, where $q_i(t)$ is the queueing delay.
- ▶ One-bit loss or no loss indication

TCP FAST: Window Control

Under normal network conditions, FAST periodically updates the congestion window based on the average RTT according to

$$w_i(t + 1) = \min\{2w, (1 - \gamma)w_i(t) + \gamma(\frac{\text{baseRTT}}{T_i(t)} w_i(t) + \alpha_i)\}$$

Where $\gamma \in (0,1]$, and α is a positive protocol parameter that determines the total number of packets queued in routers in equilibrium along the flow's path. A typical window update period is 20 ms.

Packets are transmitted in a self clocked manner, i.e., a new packet is sent out after an ACK for a previously transmitted packet arrives, which implies that the throughput $r_i(t)$ is given by

$$r_i(t) = \frac{w_i(t)}{T_i(t)}$$

Ignoring the $2w$ term, the window update rule can be written as

$$w_i(t + 1) = w_i(t) + \gamma(\alpha_i(t) - r_i(t)q_i(t)), \text{ where}$$

$$q_i(t) = T_i(t) - \text{baseRTT} \text{ is the queueing delay.}$$

Compare this with the TCP Vegas window update rule

$$w_i(t + 1) = w_i(t) + \frac{1}{T_i(t)} \operatorname{sgn}(\alpha_i(t) - r_i(t)q_i(t))$$

TCP FAST: Window Control

- ▶ While Vegas can change its window by at most 1 per RTT, window adjustment in FAST depends on the magnitude (as well as sign) of the term in brackets
- ▶ Hence FAST can adjust its window by a large amount, up or down, when the number of buffered packets is far away from its target, and a small amount when it is close.
- ▶ FAST does not react to packet losses.

$$w_i(t + 1) = w_i(t) + \gamma(\alpha_i(t) - r_i(t)q_i(t))$$

Optimization Function for TCP FAST

The equilibrium throughputs for FAST are the unique optimal vector r^* that maximizes

$$\sum_i \alpha_i \log r_i$$

Subject to the link constraint that the aggregate flow rate at any link does not exceed link capacity. Thus FAST achieves Proportional Fairness.

Note that α_i is equal to the number of flow i packets buffered in the routers in its path in steady state.

If there are N flows, the total number of packets buffered in all the routers is

$$\sum_{i=1}^N \alpha_i$$

The unique equilibrium point for FAST is

$$r_i^* = \frac{\alpha_i}{q_i^*} \quad \text{and} \quad q_i^* = T_i^* - \text{baseRTT}$$

Hence if the α_i 's are equal then theoretically each flow should obtain an equal share of the bottleneck.

Experimental Results: Intra-Protocol Fairness

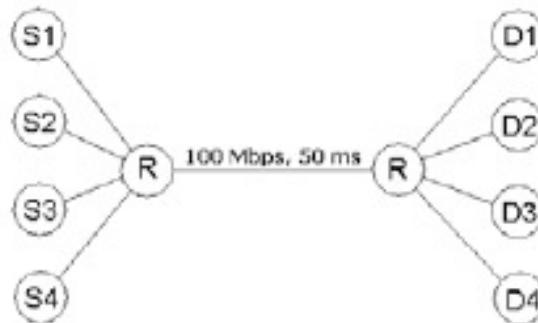


Fig. 1. The dumbbell topology used in the simulations.

For each flow, we set the parameter $\alpha_i = 200$ packets

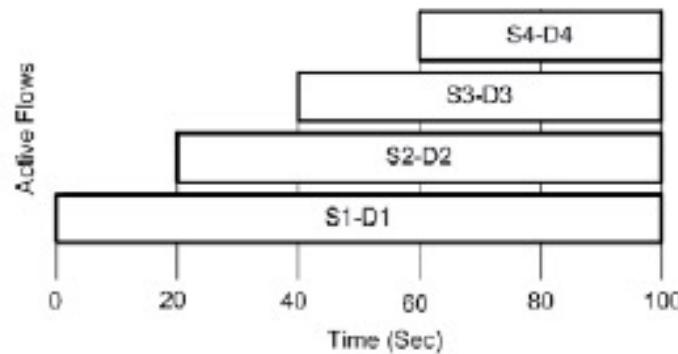
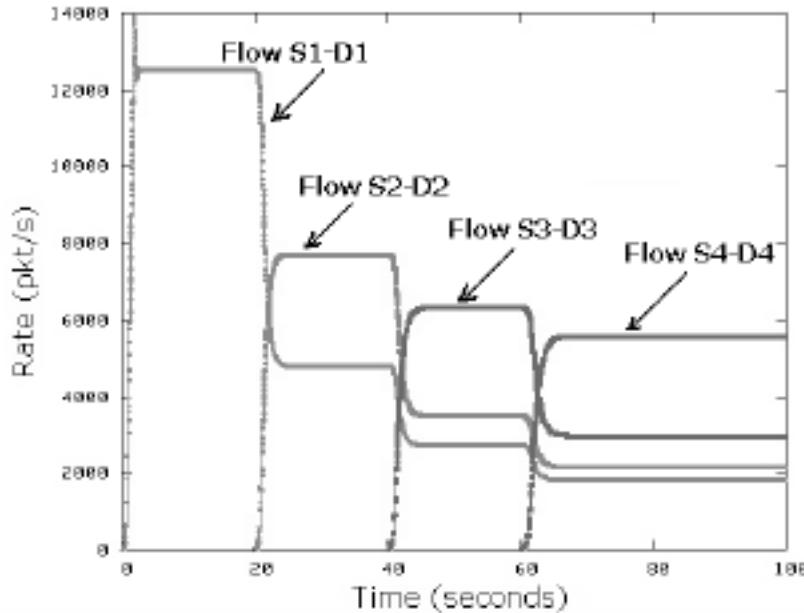


Fig. 2. The active periods of the four flows.

Experimental Results



- Note that the value of baseRTT is set by minimum RTT observed so far. Clearly this may not be an accurate estimate of the minimum RTT.
- A late joining FAST TCP flow overestimates its RTT because ALL its packets experience significant queuing delay thus its baseRTT is too high.
- Therefore it underestimates its queuing delay relative to early joining FAST TCP flows. This makes the late joining flows more aggressive thus obtaining higher throughput.

TCP COPA (2018)

Copa: Practical Delay-Based Congestion Control for the Internet

Venkat Arun and Hari Balakrishnan
M.I.T. Computer Science and Artificial Intelligence Laboratory
Email: {venkatar,hari}@mit.edu

Abstract

This paper introduces Copa, an end-to-end congestion control algorithm that uses three ideas. First, it shows that a *target rate* equal to $1/(\delta d_q)$, where d_q is the (measured) queuing delay, optimizes a natural function of throughput and delay under a Markovian packet arrival model. Second, it adjusts its congestion window in the direction of this target rate, converging quickly to the correct fair rates even in the face of significant flow churn. These two ideas enable a group of Copa flows to maintain high utilization with low queuing delay. However, when the bottleneck is shared with loss-based congestion-controlled flows that fill up buffers, Copa, like other delay-sensitive schemes, achieves low throughput. To combat this problem, Copa uses a third idea: detect the presence of buffer-filers by observing the delay evolution, and respond with additive-increase/multiplicative decrease on the δ parameter. Experimental results show that Copa outperforms Cubic (similar throughput, much lower delay, fairer with diverse RTTs), BBR and PCC (significantly fairer, lower delay), and co-exists well with Cubic unlike BBR and PCC. Copa is also robust to non-congestive loss and large bottleneck buffers, and outperforms other schemes on long-RTT paths.

1 Introduction

A good end-to-end congestion control protocol for the Internet must achieve high throughput, low queuing delay, and allocate rates to flows in a fair way. Despite three decades of work, these goals have been hard to achieve. One reason is that network technologies and applications have been continually changing. Since the deployment of Cubic [13] and Compound [32, 31] a decade ago to improve on Reno's [16] performance on high bandwidth-delay product (BDP) paths, link rates have increased significantly, wireless (with its time-varying link rates) has become common, and the Internet has become more global with terrestrial paths exhibiting higher round-trip times (RTTs) than before. Faster link rates mean that many flows start and stop quicker, increasing the level of flow churn, but the prevalence of video streaming and large bulk transfers (e.g., file sharing and backups) means that these long flows must co-exist with short ones whose objectives are different (high throughput versus low flow completion

time or low interactive delay). Larger BDPs exacerbate the “bufferbloat” problem. A more global Internet leads to flows with very different propagation delays sharing a bottleneck (exacerbating the RTT-unfairness exhibited by many current protocols).

At the same time, application providers and users have become far more sensitive to performance, with notions of “quality of experience” for real-time and streaming media, and various metrics to measure Web performance being developed. Many companies have invested substantial amounts of money to improve network and application performance. Thus, the performance of congestion control algorithms, which are at the core of the transport protocols used to deliver data on the Internet, is important to understand and improve.

Congestion control research has evolved in multiple threads. One thread, starting from Reno, and extending to Cubic and Compound relies on packet loss (or ECN) as the fundamental congestion signal. Because these schemes fill up network buffers, they achieve high throughput at the expense of queuing delay, which makes it difficult for interactive or Web-like applications to achieve good performance when long-running flows also share the bottleneck. To address this problem, schemes like Vegas [4] and FAST [34] use delay, rather than loss, as the congestion signal. Unfortunately, these schemes are prone to overestimate delay due to ACK compression and network jitter, and under-utilize the link as a result. Moreover, when run with concurrent loss-based algorithms, these methods achieve poor throughput because loss-based methods must fill buffers to elicit a congestion signal.

A third thread of research, starting about ten years ago, has focused on important special cases of network environments or workloads, rather than strive for generality. The past few years have seen new congestion control methods for datacenters [1, 2, 3, 29], cellular networks [36, 38], Web applications [9], video streaming [10, 20], vehicular Wi-Fi [8, 21], and more. The performance of special-purpose congestion control methods is often significantly better than prior general-purpose schemes.

A fourth, and most recent, thread of end-to-end congestion control research has argued that the space of congestion control signals and actions is too complicated for human engineering, and that algorithms

COPA: Objective

COPA was designed with the objective of solving three of the four issues that were pointed out for delay based algorithms:

- ▶ Ramping up window for high speed networks: COPA includes a mechanism for using higher window increments for high speed networks.
- ▶ Competing with loss based systems: COPA is able to detect if there are competing flows at a node that are loss based, and it changes its window control algorithm appropriately.
- ▶ Accurately measuring the minimum round trip delay T: The algorithm guarantees that the queue drains completely periodically, which helps to measure T.

COPA: Approach

- ▶ Start with an objective function to optimize. The objective function combines a flow's average throughput r , and packet delay (minus propagation delay) q

$$U = \log r - \delta \log q$$

The goal is for each sender to maximize its U .

- ▶ Here δ determines how much to weigh delay compared to throughput; a larger δ signifies that lower packet delays are preferable.
- ▶ Under certain simplified (but reasonable) modeling assumptions of packet arrivals, the steady-state sending rate (in packets per second) that maximizes U is

$$r = \frac{1}{\delta q^*}$$

Where q^* is the mean per packet queueing delay.

- ▶ When every sender transmits at this rate, a unique, socially-acceptable Nash equilibrium is attained.

COPA: Approach (cont)

$$U = \log r - \delta \log q$$

- ▶ Where did this Objective Function come from? Recall that the Objective Function that TCP Vegas optimizes is

$$U_i(r_i) = w_i \log r_i$$

- ▶ We arrived at this function by starting from the window control rules and then deriving a function that these rules optimize.
- ▶ COPA reverses this: We start with an Objective Function and then try to come up with the window dynamics that optimize this function.
- ▶ This is done in two steps:
 - Step 1: Use the Objective Function to obtain a formula for the equilibrium throughput.
 - Step 2: Derive the window control rules that are compatible with this throughput formula.

COPA Step 1: Target Rate

$$\dot{r} = \frac{1}{\delta q^*}$$

- ▶ This rate is used as the target rate for a Copa sender. The sender estimates the queuing delay using its RTT observations, and moves quickly toward hovering near this target rate.
- ▶ This mechanism induces a property that the queue is regularly almost flushed every 5RTT, which helps all endpoints get a correct estimate of the minimum RTT.
- ▶ Note that since COPA does not react to packet losses, it may unfairly hog the bandwidth when competing with buffer filling type flows (which happens with BBR).
In order to prevent this, Copa mimics an AIMD window–update rule when it observes that the bottleneck queues rarely empty (similar to Yeah TCP).

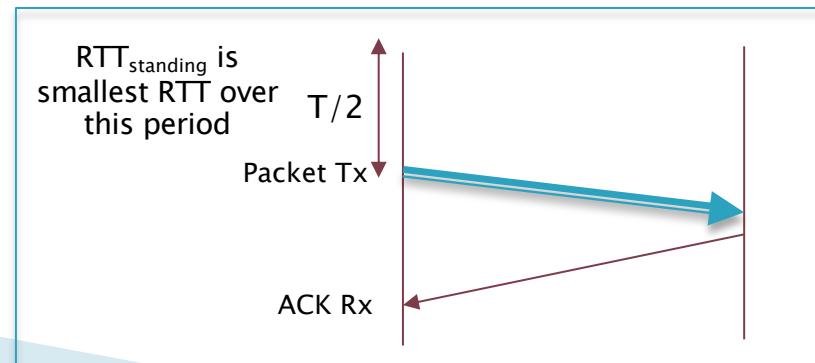
COPA Step 2: cwnd Updating Rule

- ▶ Copa uses a congestion window, cwnd, which upper-bounds the number of in-flight packets.
- ▶ On every ACK received, the sender estimates the current rate $r = \text{cwnd}/\text{RTT}_{\text{standing}}$, where $\text{RTT}_{\text{standing}}$ is the smallest RTT observed over a recent time-window, τ (this corresponds to a standing-queue at the node)

$$\tau = \frac{T}{2}$$

Where T is current value of the smoothed RTT estimate.

- ▶ The reason for using the smallest RTT in the recent $\tau = \text{srtt}/2$ duration, rather than the latest RTT sample, is for robustness in the face of ACK compression and network jitter, which increase the RTT and can confuse the sender into believing that a longer RTT is due to queueing on the forward data path.



COPA Step 2: cwnd Updating Rule

- ▶ The source calculates the target rate using $r = \frac{1}{\delta q^*}$, where

$$q^* = \text{RTT}_{\text{standing}} - \text{baseRTT} \quad \text{---(1)}$$

where baseRTT is the smallest RTT observed over a long period of time. COPA uses the smaller of 10 seconds and the time since the flow started for this period.

- ▶ If the current rate exceeds the target rate, the sender reduces cwnd; otherwise, it increases cwnd.
- ▶ To avoid packet bursts, the sender paces packets at a rate of $2 * \text{cwnd} / \text{RTT}_{\text{standing}}$ packets per second.

$\frac{1}{\delta}$ is in units of MTU sized packets

Window Update On Every ACK Arrival

1. Update the queuing delay q^* and srtt using the standard TCP exponentially weighted moving average estimator.

2. Set target rate $r_t = \frac{1}{\delta q^*}$

3. If the current rate

$$r = \text{cwnd}/\text{RTT}_{\text{standing}} \leq r_t , \text{ then}$$

$$\text{cwnd} = \text{cwnd} + v/(\delta \cdot \text{cwnd}),$$

where v is a velocity parameter" (defined in the next step).

Otherwise,

$$\text{cwnd} = \text{cwnd} - v/(\delta \cdot \text{cwnd})$$

Over 1 RTT, the change in cwnd is thus $\approx v/\delta$ packets.

$$q^* = \text{RTT}_{\text{standing}} - \text{baseRTT}$$

Window Update (cont)

1. The velocity parameter, v , speeds-up convergence. It is initialized to 1. Once per window, the sender compares the current cwnd to the cwnd value at the time that the latest acknowledged packet was sent (i.e., cwnd at the start of the current window).
 - o If the current cwnd is larger, then set direction to "up"; if it is smaller, then set direction to "down".
 - o If direction is the same as in the previous window, then double v . If not, then reset v to 1. However, start doubling v only after the direction has remained the same for three RTTs.
2. When a flow starts, Copa performs slow-start where cwnd doubles once per RTT until r exceeds r_t . While the velocity parameter also allows an exponential increase, the constants are smaller.

COPA in Steady State

- ▶ Note that: Sending Rate < Target Rate implies that

$$\frac{cwnd}{RTT_{standing}} < \frac{1}{\delta(RTT_{standing} - RTT_{min})}$$

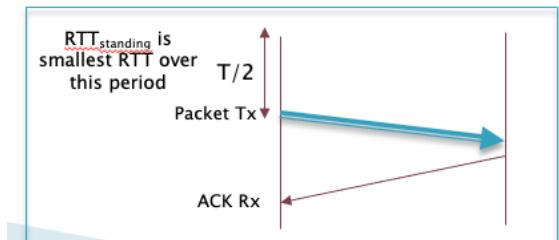
Which is the same as

$$\frac{cwnd}{RTT_{standing}} (RTT_{standing} - RTT_{min}) < \frac{1}{\delta}$$

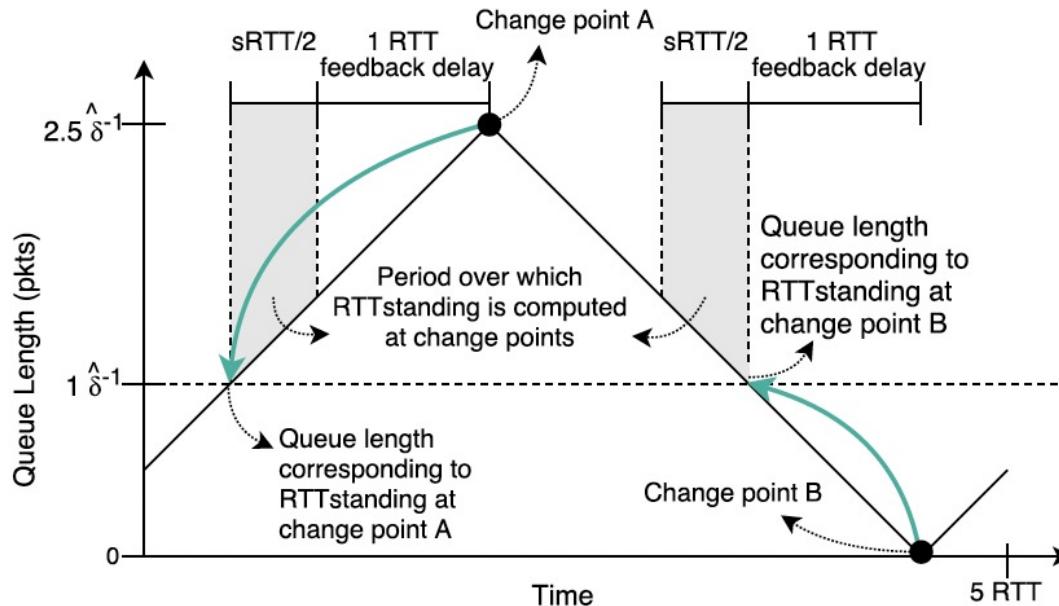
i.e, the queue size at the bottleneck is less than $\frac{1}{\delta}$.

- ▶ Sending Rate > Target Rate coincides with the event that the bottleneck queue size has exceeded $\frac{1}{\delta}$.
- ▶ However it takes $1.5T$ for this information to be incorporated into the rate calculation.
 - This is because of the way $RTT_{standing}$ is computed

$RTT_{standing}$ is the smallest RTT observed over a recent time-window

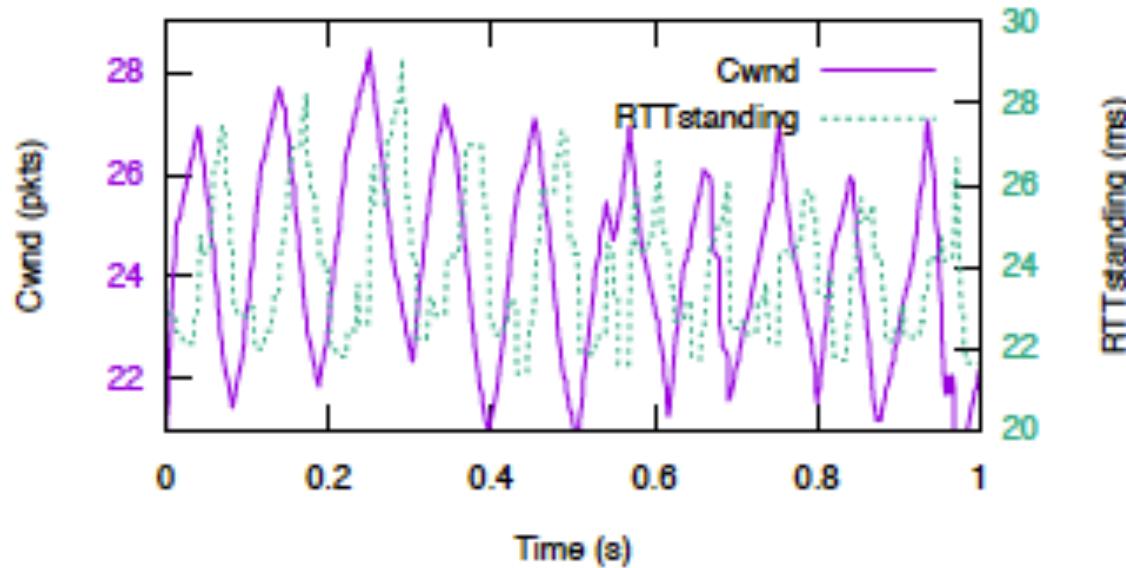


Steady State Oscillations



- Period of oscillations = $5T$
- The queue size varies from zero to $\frac{2.5}{\delta}$
- The bottleneck queue empties at the start and end of a period.
- With multiple flows with same propagation delay, their oscillations synchronize with $\hat{\delta} = \left(\sum_i \frac{1}{\delta_i} \right)^{-1}$
- Queue emptying is a critical property since it helps to measure RTT_{min} accurately and it also facilitates intra-protocol fairness.

COPA Window Evolution



The period of oscillation is 5RTT and amplitude is 5 packets (since $\delta = 0.5$).

Competing with Buffer Based Algorithms

- ▶ If Copa seeks to maintain low queuing delays; without modification, it will lose to buffer-filling schemes.
- ▶ Modifications: There are two distinct modes of operation for Copa:
 1. The **default mode** where $\delta = 0.5$, and
 2. A **competitive mode** where δ is adjusted dynamically to match the aggressiveness of typical buffer-filling schemes.
- ▶ Copa switches between these modes depending on whether or not it detects a competing long-running buffer-filling scheme. The detector exploits a key Copa property that the queue is empty at least once every 5 RTT when only Copa flows with similar RTTs share the bottleneck. Hence if the sender sees a nearly empty" queue in the last 5 RTTs, it remains in the default mode; otherwise, it switches to competitive mode.
- ▶ We estimate nearly empty" as any queuing delay lower than 10% of the rate oscillations in the last four RTTs; i.e., $q < 0.1(\text{RTT}_{\max} - \text{RTT}_{\min})$.
- ▶ In competitive mode the sender varies $1/\delta$ according to whatever buffer-filling algorithm one wishes to emulate (e.g., NewReno, Cubic, etc.). COPA performs AIMD on $1/\delta$ based on packet success or loss,

Over 1 RTT the change in cwnd is $1/\delta$ packets

COPA Performance

- ▶ COPA performance:
 - COPA exhibits higher intra-protocol fairness compared to Cubic and BBR.
 - COPA achieved as much throughput and 2–10x lower queueing delays compared to Cubic and BBR.
 - In datacenter network simulations, on a web search workload trace drawn from datacenter network, Copa achieved a $> 5\times$ reduction in flow completion time for short flows over DCTCP. It achieved similar performance for long flows.
 - In experiments on an emulated satellite path, Copa achieved nearly full link utilization with a median queuing delay of only 1 ms. BBR obtained 50% link utilization. Both Cubic and Vegas obtained $< 4\%$ utilization.
 - COPA also exhibits better RTT fairness.

COPA Performance

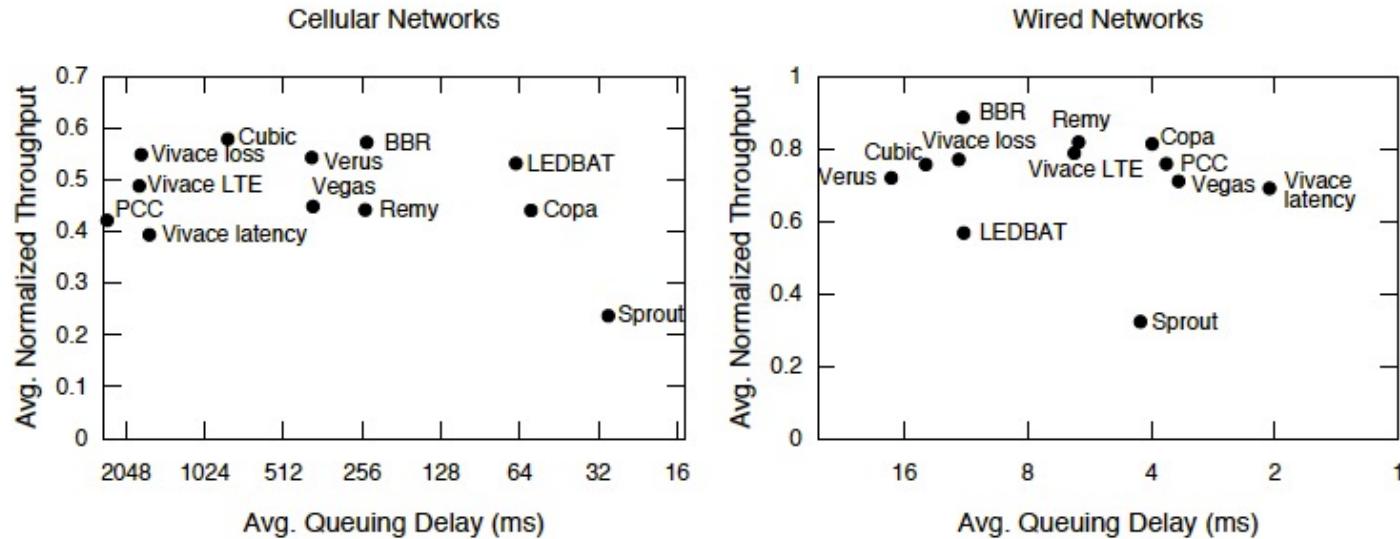


Figure 5: Real-world experiments on Pantheon paths: Average normalized throughput vs. queuing delay achieved by various congestion control algorithms under two different types of Internet connections. Each type is averaged over several runs over 6 Internet paths. Note the very different axis ranges in the two graphs. The x-axis is flipped and in log scale. Copa achieves consistently low queueing delay and high throughput in both types of networks. Note that schemes such as Sprout, Verus, and Vivace LTE are designed specifically for cellular networks. Other schemes that do well in one type of network don't do well on the other type. On wired Ethernet paths, Copa's delays are 10 \times lower than BBR and Cubic, with only a modest mean throughput reduction.

COPA Performance with Link Loss

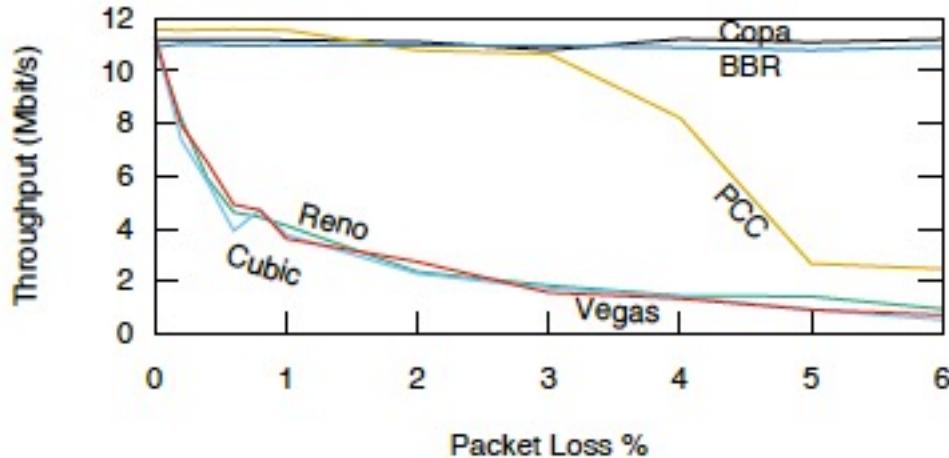


Figure 7: Performance of various schemes in the presence of stochastic packet loss over a 12 Mbit/s emulated link with a 50 ms RTT.

TIMELY

TIMELY: RTT-based Congestion Control for the Datacenter

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

Google, Inc.

ABSTRACT

Datacenter transports aim to deliver low latency messaging together with high throughput. We show that simple packet delay, measured as round-trip times at hosts, is an effective congestion signal without the need for switch feedback. First, we show that advances in NIC hardware have made RTT measurement possible with microsecond accuracy, and that these RTTs are sufficient to estimate switch queuing. Then we describe how TIMELY can adjust transmission rates using RTT gradients to keep packet latency low while delivering high bandwidth. We implement our design in host software running over NICs with OS-bypass capabilities. We show using experiments with up to hundreds of machines on a Clos network topology that it provides excellent performance: turning on TIMELY for OS-bypass messaging over a fabric with PFC lowers 99 percentile tail latency by 9X while maintaining near line-rate throughput. Our system also outperforms DCTCP running in an optimized kernel, reducing tail latency by 13X. To the best of our knowledge, TIMELY is the first delay-based congestion control protocol for use in the datacenter, and it achieves its results despite having an order of magnitude fewer RTT signals (due to NIC offload) than earlier delay-based schemes such as Vegas.

CCS Concepts

•Networks → Transport protocols;

Keywords

datacenter transport; delay-based congestion control; OS-bypass; RDMA

^{*}Work done while at Google

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

© 2015 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-3542-3/15/08

DOI: <http://dx.doi.org/10.1145/2785956.2787510>

1. INTRODUCTION

Datacenter networks run tightly-coupled computing tasks that must be responsive to users, e.g., thousands of back-end computers may exchange information to serve a user request, and all of the transfers must complete quickly enough to let the complete response to be satisfied within 100 ms [24]. To meet these requirements, datacenter transports must simultaneously deliver high bandwidth (\gg Gbps) and utilization at low latency (\ll msec), even though these aspects of performance are at odds. Consistently low latency matters because even a small fraction of late operations can cause a ripple effect that degrades application performance [21]. As a result, datacenter transports must strictly bound latency and packet loss.

Since traditional loss-based transports do not meet these strict requirements, new datacenter transports [10, 18, 30, 35, 37, 47], take advantage of network support to signal the onset of congestion (e.g., DCTCP [35] and its successors use ECN), introduce flow abstractions to minimize completion latency, cede scheduling to a central controller, and more. However, in this work we take a step back in search of a simpler, immediately deployable design.

The crux of our search is the congestion signal. An ideal signal would have several properties. It would be fine-grained and timely to quickly inform senders about the extent of congestion. It would be discriminative enough to work in complex environments with multiple traffic classes. And, it would be easy to deploy.

Surprisingly, we find that a well-known signal, properly adapted, can meet all of our goals: delay in the form of RTT measurements. RTT is a fine-grained measure of congestion that comes with every acknowledgment. It effectively supports multiple traffic classes by providing an inflated measure for lower-priority transfers that wait behind higher-priority ones. Further, it requires no support from network switches.

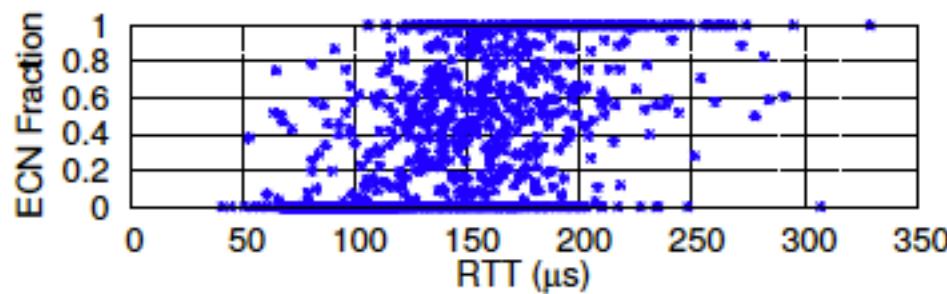
Delay has been explored in the wide-area Internet since at least TCP Vegas [16], and some modern TCP variants use delay estimates [44, 46]. But this use of delay has not been without problems. Delay-based schemes tend to compete poorly with more aggressive, loss-based schemes, and delay

TIMELY Congestion Control

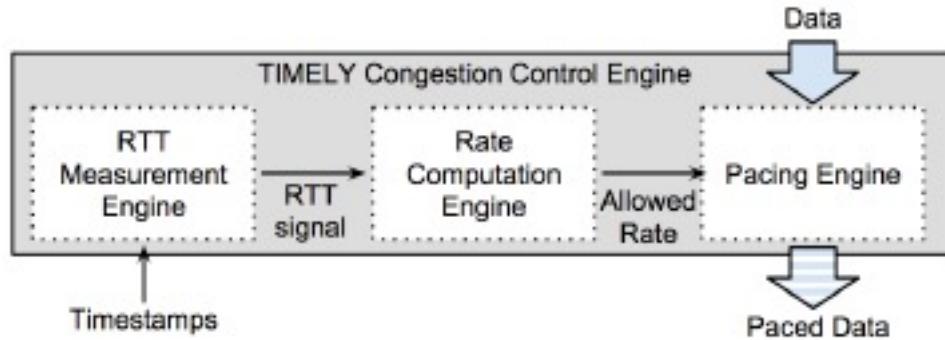
- ▶ Specialized for Data Center environments
- ▶ Uses very precise measurements for round trip latency, made possible by advances in NIC Card technology
- ▶ Uses a Rate Based congestion control algorithm
- ▶ Uses a combination of **Delay AND Delay Gradients** to keep queue size within a tight range.
- ▶ Requires no switch support, purely end-to-end based

Delay based Algorithms in the Data Center

- ▶ Delay had not been used as a congestion signal in the datacenter because datacenter RTTs are difficult to measure at microsecond granularity. This level of precision is easily overwhelmed by host delays such as interrupt processing for acknowledgments.
- ▶ However recent NIC advances do allow datacenter RTTs to be measured with sufficient precision. They provide hardware support for high-quality timestamping of packet events plus hardware-generated ACKs that remove unpredictable host response delays.
- ▶ Using RTTs provides richer and faster information about the state of network switches than explicit network switch signals such as ECN marks. The ECN signal does not correlate well with RTT and hence with the amount of queuing.



TIMELY Protocol



TIMELY has 3 components

1. RTT measurement to monitor the network for congestion
2. A computation engine that converts RTT signals into target sending rates
3. A control engine that inserts delays between segments to achieve the target rate.

RTT Measurement Engine

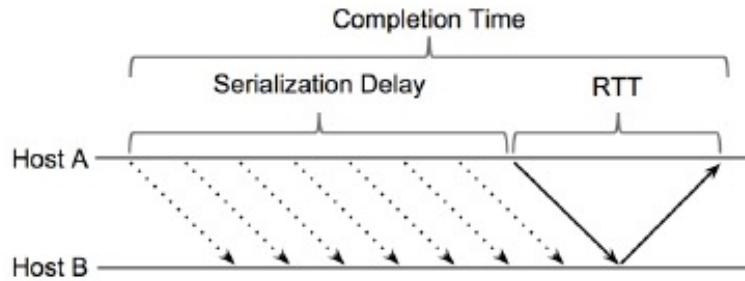


Figure 7: Finding RTT from completion time.

- ▶ A segment consisting of multiple packets is sent as a single burst and then ACKed as a unit by the receiver. A completion event is generated upon receiving an ACK for a segment of data and includes the ACK receive time.
- ▶ The time from when the first packet is sent (t_{send}) until the ACK is received ($t_{completion}$) is defined as the completion time.

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

↑ ↓
Propagation Serialization
Delay Delay

TIMELY Congestion Control

- ▶ The rate computation engine runs the congestion control algorithm upon each completion event, and outputs an updated target rate for the flow.
- ▶ TIMELY does not try to control the queueing delay directly since it has been shown that it is not possible to control the queue size when it is shorter in time than the control loop delay.
- ▶ TIMELY's congestion controller achieves low latencies by reacting to the **delay gradient** or derivative of the queuing with respect to time, instead of trying to maintain a standing queue.
 - This is possible because we can accurately measure differences in RTTs that indicate changes in queuing delay.
 - A positive delay gradient due to increasing RTTs indicates a rising queue, while a negative gradient indicates a receding queue.
 - By using the gradient, we can react to queue growth without waiting for a standing queue to form – a strategy that helps us achieve low latencies.

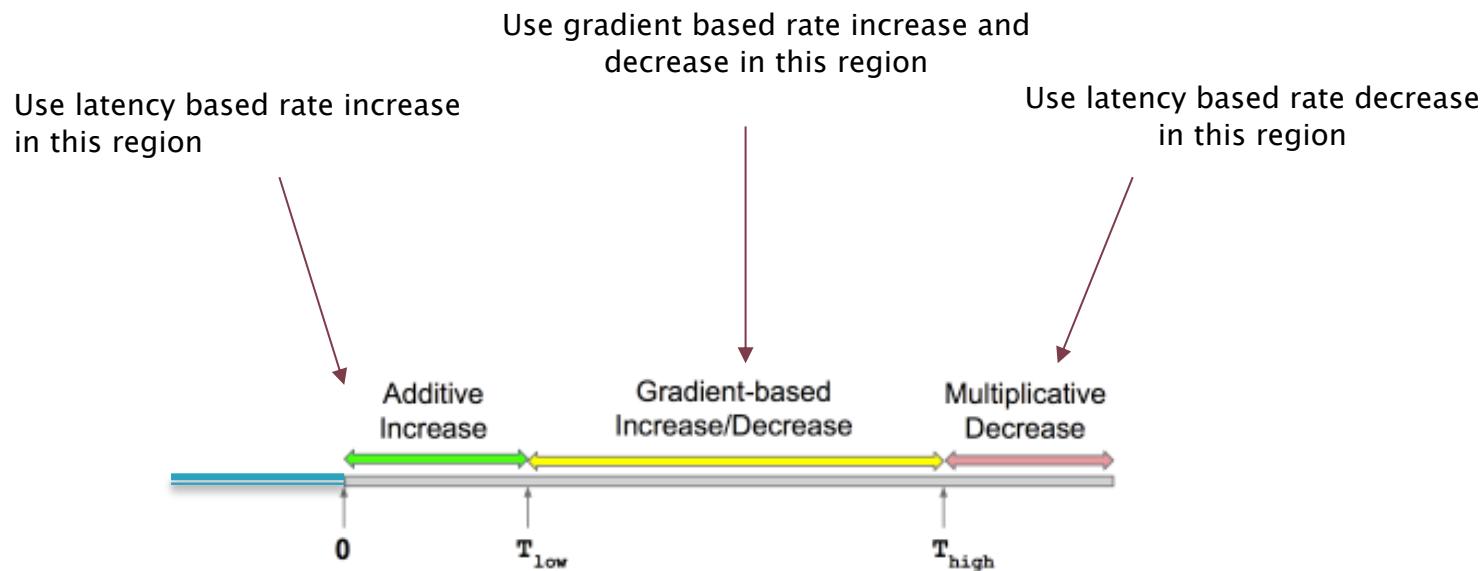
Delay Gradients

- ▶ Delay gradient is a proxy for the rate mismatch at the bottleneck queue. RCP, XCP etc have shown that explicit feedback on the rate mismatch has better stability and convergence properties than explicit feedback based only on queue sizes.
- ▶ We denote the queuing delay through the bottleneck queue by $q(t)$. If $y(t) > C$, the rate at which the queue builds up is $(y(t) - C)$. Since queued data drains at a rate C , the queuing delay gradient is given by

$$\frac{(y(t) - C)}{C} = \frac{dq(t)}{dt} = \frac{d(RTT)}{dt}.$$

- ▶ Hence, the delay gradient measured through RTT signals acts as an indicator for the rate mismatch at the bottleneck.

Timely Congestion Control



- The T_{low} and T_{high} thresholds effectively bring the delay within a target range and play a role similar to the target queue occupancy in many AQM schemes.
- Using the delay gradient improves stability and helps keep the latency within the target range $[T_{low}, T_{high}]$.

Timely Congestion Control

Algorithm 1: TIMELY congestion control.

Data: new_rtt

Result: Enforced rate

new_rtt_diff = new_rtt - prev_rtt ;

prev_rtt = new_rtt ;

rtt_diff = $(1 - \alpha) \cdot rtt_diff + \alpha \cdot new_rtt_diff$;

▷ α : EWMA weight parameter

normalized_gradient = rtt_diff / minRTT ;

if $new_rtt < T_{low}$ then

 rate \leftarrow rate + δ ;

 ▷ δ : additive increment step

 return;

if $new_rtt > T_{high}$ then

 rate \leftarrow rate $\cdot (1 - \beta \cdot (1 - \frac{T_{high}}{new_rtt}))$;

 ▷ β : multiplicative decrement factor

 return;

if $normalized_gradient \leq 0$ then

 rate \leftarrow rate + $N \cdot \delta$;

 ▷ $N = 5$ if gradient < 0 for five completion events

 (HAI mode); otherwise $N = 1$

else

 rate \leftarrow rate $\cdot (1 - \beta \cdot normalized_gradient)$

Filtered estimate of
Delay Gradient

If the gradient is negative or equals zero,
the network can keep up with the aggregate
incoming rate, and therefore
there is room for a higher rate.

When the gradient is positive, the
total sending rate is greater than
network capacity. Hence,
TIMELY performs a multiplicative
rate decrement, scaled
by the gradient factor:

Comparison with DCTCP

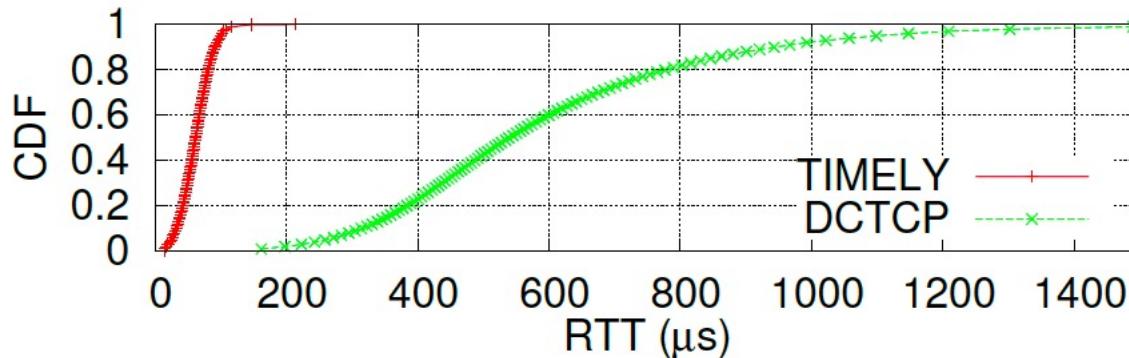


Figure 14: CDF of RTT distribution

- TIMELY keeps the average end-to-end RTT 10X lower than DCTCP (60 s vs. 600 s).
- More significantly, the tail latency drops by almost 13X (116 s vs. 1490 s).

Issues with Delay Bounding Schemes (2022)

Starvation in End-to-End Congestion Control

Venkat Arun, Mohammad Alizadeh, Hari Balakrishnan

{venkatar, alizadeh, hari}@csail.mit.edu

MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA

ABSTRACT

To overcome weaknesses in traditional loss-based congestion control algorithms (CCAs), researchers have developed and deployed several delay-bounding CCAs that achieve high utilization without bloating delays (e.g., Vegas, FAST, BBR, PCC, Copia, etc.). When run on a path with a fixed bottleneck rate, these CCAs converge to a small delay range in equilibrium. This paper proves a surprising result: although designed to achieve reasonable inter-flow fairness, current methods to develop delay-bounding CCAs cannot always avoid starvation, an extreme form of unfairness. Starvation may occur when such a CCA runs on paths where non-congestive network delay variations due to real-world factors such as ACK aggregation and end-host scheduling exceed double the delay range that the CCA converges to in equilibrium. We provide experimental evidence for this result for BBR, FCC Vivace, and Copia with a link emulator. We discuss the implications of this result and posit that to guarantee no starvation an efficient delay-bounding CCA should design for a certain amount of non-congestive jitter and ensure that its equilibrium delay oscillations are at least one-half of this jitter.

CCS CONCEPTS

• Networks → Transport protocols; Protocol correctness;

KEYWORDS

Congestion Control, Delay-Convergence, Starvation

ACM Reference Format:

Venkat Arun, Mohammad Alizadeh, Hari Balakrishnan {venkatar, alizadeh, hari}@csail.mit.edu. *MIT Computer Science and Artificial Intelligence Lab, Cambridge, MA, 2022*. Starvation in End-to-End Congestion Control. In *ACM SIGCOMM 2022 Conference (SIGCOMM '22)*, August 22–26, 2022, Amsterdam, Netherlands. ACM, New York, NY, USA, 16 pages. <https://doi.org/10.1145/3544216.3544223>

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

SIGCOMM '22, August 22–26, 2022, Amsterdam, Netherlands

© 2022 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-9420-8/22/08.

<https://doi.org/10.1145/3544216.3544223>

1 INTRODUCTION

With the rise of interactive and real-time applications, the widespread recognition of bufferbloat as a significant problem [17], and increasing user expectations for high quality-of-experience, the networking community has developed a variety of *delay-bounding* congestion control algorithms (CCAs) for applications to use on the Internet. Unlike CCAs such as Reno/NewReno [22, 24], Cubic [20], and Compound [41], delay-bounding CCAs do not keep increasing their congestion window (*cwnd*) until they experience packet loss or receive acknowledgments (ACKs) with explicit congestion notification (ECN) bits [14]. Instead they use the measured round-trip time (RTT) and other factors (e.g., rate estimates) to set *cwnd*. They aim to achieve high utilization without bloating delays, and are also more efficient in the face of some packet loss.

The development of delay-bounding CCAs for the Internet began with Vegas 1994 [6], followed by FAST [44], but largely stagnated because of the inability of these schemes to obtain good throughput when competing with buffer-filling CCAs such as Reno/NewReno and especially Cubic. Over the past ten years, however, delay-bounding CCAs have experienced a resurgence with several proposals that overcome these issues, including Sprout [46], Remy [45], BBR [8, 10], PCC [12, 13], Copia [3], and Verus [48]. Some of these schemes are widely deployed, most notably BBR, which is now the CCA used by many popular Internet sites.

In this paper we study how multiple flows running a given delay-bounding CCA share bandwidth. We start by noting that many CCAs share a common property. On ideal network paths with a constant bottleneck rate and propagation delay, they converge to a small delay range and oscillate within that range. The mean queuing delay range experienced at equilibrium is either constant (no matter what the bottleneck rate) or a decreasing function of the bottleneck rate (e.g., the CCA maintains a certain number of enqueued packets).

We show that this convergence property has an important consequence. Because most CCAs attempt to work across many orders of magnitude of rates, they *must* map a large rate range into a small delay range.¹ Thus, even small changes in estimated queuing delay would induce enormous changes

¹Unless they are also willing to let the delay bound grow with the rate, which is not interesting in practice.

Objective

- ▶ The paper focuses on how multiple flows running a given delay-bounding CCA share bandwidth.
- ▶ The problem the authors identified is the following: Because most CCAs attempt to work across many orders of magnitude of rates, **they must map a large rate range into a small delay range**. Thus, even small changes in estimated queueing delay would induce enormous changes in the inferred rate.
- ▶ Can perfect measurements solve the problem:
 - No, since there is no way to precisely measure congestive queueing delays. The reason is that non-congestive (i.e., non-bottleneck-queueing) contributions to network delay are rarely constant in practice, and hard to separate from congestion.
 - Due to effects such as ACK aggregation, delayed ACKs, end-host operating system or thread scheduling, token bucket filters, hardware offload timing variations, etc., packets experience jitter on network paths unrelated to bottleneck queueing.
- ▶ Hence they focus on understanding the effects of non-congestive jitter on delay-bounding CCAs, expecting some degree of unfairness and hoping to quantify it as a function of the delay jitter.

Main Results

- ▶ Efficient delay-bounding CCAs that converge in steady state to a bounded delay range are not merely unfair, but cannot avoid starvation.
- ▶ When two flows using the same CCA share a bottleneck link, if the non-congestive delay variations exceed double the difference between the maximum and minimum queueing delay at equilibrium, then there are patterns of non-congestive delay where one flow will get arbitrarily low throughput compared to the other.
- ▶ Congestion Control Algorithms can target at most two out of the following three properties:
 1. High throughput,
 2. Convergence to a small and bounded delay range, and
 3. No starvation.
- ▶ Are we are doomed to choose between bounding delays and avoiding starvation?
 - We might be able to achieve both desirable goals by being explicit about non-congestive delays in CCA design, ensuring that the CCA's delay variations in equilibrium are at least half as large as the non-congestive jitter expected along a path.
 - If that is not the case, then the results prove that starvation is inevitable.

Results (cont)

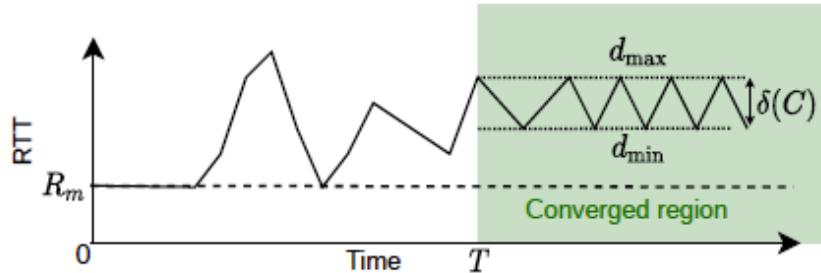
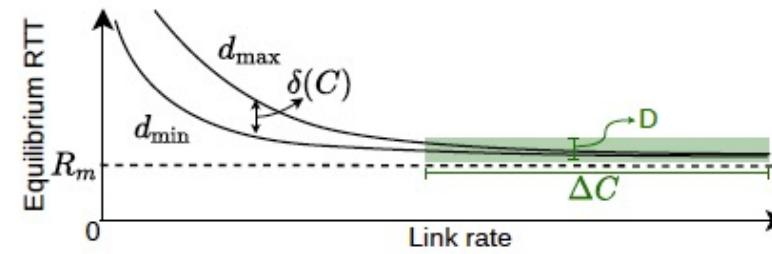


Figure 1: Ideal-path behavior of a hypothetical delay-convergent CCA.

- CCAs that ensure a smaller δ_{\max} and hence are “more convergent” are more susceptible to starvation.
- Starvation can occur if the delay ambiguity caused by non-congestive jitter delay is $> 2\delta_{\max}$.
- For many CCAs, δ_{\max} is small because they strive to keep delay variations small compared to R_m . Hence even a little ambiguity can cause starvation.



Variation of $\delta (C)$ with C

Results (cont)

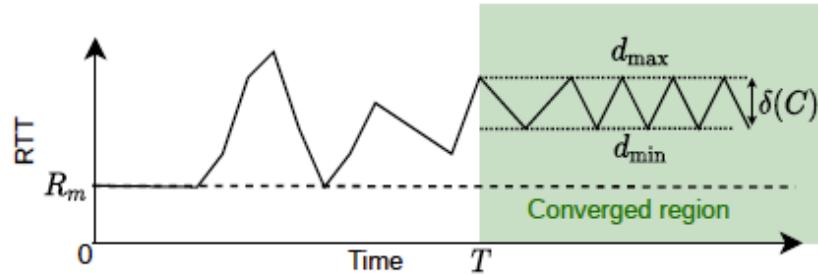
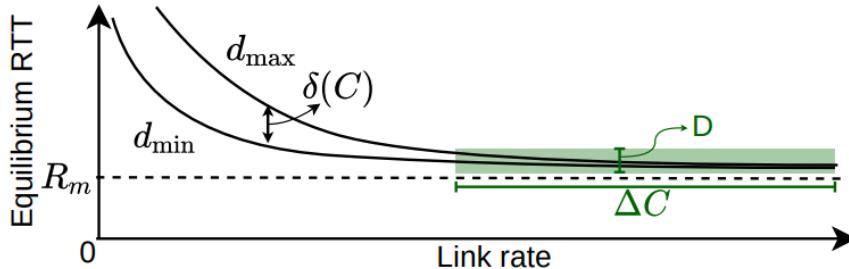


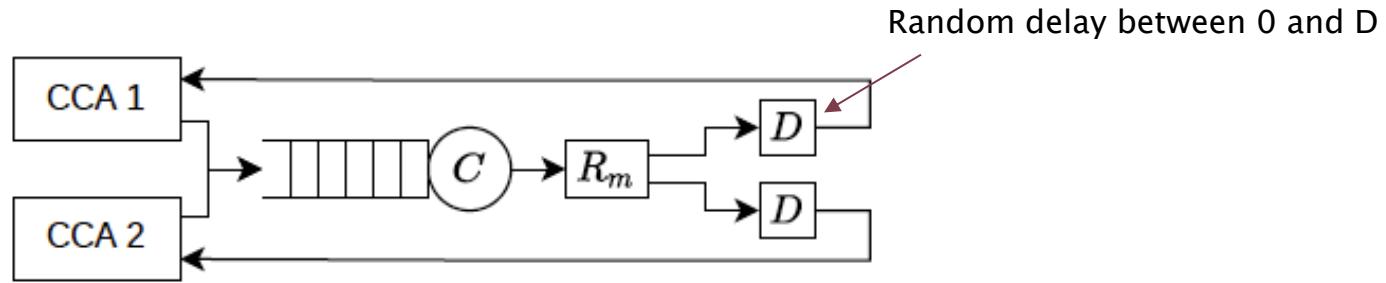
Figure 1: Ideal-path behavior of a hypothetical delay-convergent CCA.

- $\delta(C)$ is 0 for Vegas, FAST, and BBR in cwnd limited mode
- $4\alpha/C$ for Copa, where α is the packet size
- $R_m/4$ for BBR in pacing mode
- $R_m/20$ for PCC Vivace
- ALL these protocols suffer from starvation even in simple two-flow scenarios with small non-congestive jitter



Variation of delays with link rate for a fixed R_m

Network Model



- Non-congestive delays create ambiguity about the amount of queueing delay at the bottleneck.
- For example, when a CCA measures an RTT increase of $\Delta d=5$ it only knows that the increase due to congestion is between $\Delta d-D$ and Δd .
- This ambiguity creates confusion at the sender. If the delay observed is truly caused by congestion, the CCA should reduce its rate, but if it does so and the delay turns out to be non-congestive, it may under-utilize the network.
- When flows sharing a link estimate the queueing delay differently, they may send at unequal rates, which can cause starvation

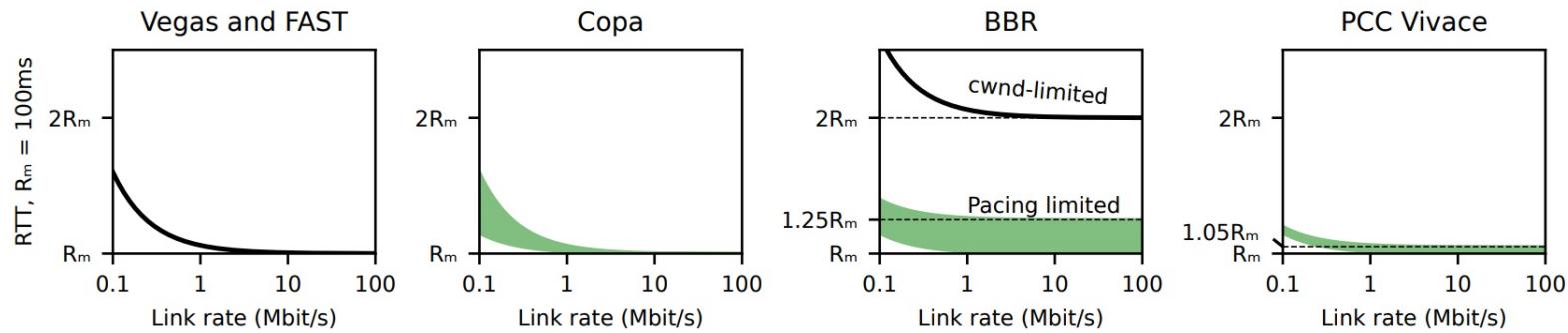
Example:

- Consider a CCA such as Vegas or FAST that seeks to maintain α packets in the queue per flow in equilibrium. α is a parameter of the algorithm (e.g., $\alpha = 4$ packets).
- Over an ideal path, once the CCA hits this target, its rate stabilizes and the queue length never changes.
- Hence $\delta_{max} = 0$ and the RTT is $R_m + \alpha/C$

An Example

- Consider a CCA such as Vegas or FAST that seeks to maintain α packets in the queue per flow in equilibrium where α is a parameter of the algorithm (e.g., $\alpha = 4$ packets).
- Over an ideal path, once the CCA hits this target, its rate stabilizes and the queue length never changes. Hence $\delta_{max} = 0$ and the RTT is $R_m + \alpha/C$
- To achieve this equilibrium, the CCA must measure the queueing delay with high precision. For example, with $\alpha = 4$ and a packet size of 1500 bytes, $\alpha/C = 0.5$ ms for $C = 96$ Mbit/s and 0.05 ms for $C = 960$ Mbit/s.
- Thus a difference of only 0.45 ms in the estimated queueing delay can cause the CCA to vary its sending rate by 10×!
- Therefore, if the delay measurement ambiguity exceeds this amount, it can easily cause severe unfairness. Delay jitter of tens of milliseconds occur on Wi-Fi and cellular links.
- Any delay-convergent CCA that seeks to bound delay variation below the level of jitter (delay ambiguity) of the network will suffer from the same problem.
- The fundamental issue is that very different link rates are consistent with similar delay measurements. When different flows experience different non-congestive delays, they can infer very different link rates, causing unfairness and starvation.

Delay Variations for Some CCAs



To Avoid Starvation

1. To utilize the link efficiently, a CCA must maintain a queue that is larger than the non-congestive delay on the path;
2. This alone is not enough to avoid starvation, but in addition the variation in the queueing delay in steady state must also be greater than one-half of the delay jitter;
3. If we have a prior upper bound on sending rate, we may be able to avoid starvation while also reducing the queueing delay variation.

Loss based algorithms actually satisfy these requirements, but the tradeoff is that delay is no longer small or bounded