Copa: Practical Delay-Based Congestion Control for the Internet

Venkat Arun and Hari Balakrishnan, ***MIT CSAIL***

https://www.usenix.org/conference/nsdi18/presentation/arun

This paper is included in the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI '18).

April 9–11, 2018 • Renton, WA, USA

ISBN 978-1-931971-43-0

Open access to the Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation is sponsored by USENIX.

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) queueing 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-fillers 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 queueing 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 queueing 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

can produce better actions than humans. Work in this thread includes Remy [30, 35], PCC [6], and Vivace [7]. These approaches define an objective function to guide the process of coming up with the set of online actions (e.g., on every ACK, or periodically) that will optimize the specified function. Remy performs this optimization offline, producing rules that map observed congestion signals to sender actions. PCC and Vivace perform online optimizations.

In many scenarios these objective-optimization methods outperform the more traditional window-update schemes [6, 35]. Their drawback, however, is that the online rules executed at runtime are much more complex and hard for humans to reason about (for example, a typical Remy controller has over 200 rules). A scheme that uses online optimization requires the ability to measure the factors that go into the objective function, which may take time to obtain; for example, PCC's default objective function incorporates the packet loss rate, but a network running at a low packet loss rate (a desirable situation) will require considerable time to estimate.

We ask whether it is possible to develop a congestion control algorithm that achieves the goals of high throughput, low queueing delay, and fair rate allocations, but which is also *simple* to understand and is *general* in its applicability to a wide range of environments and workloads, and that *performs at least as well* as the best prior schemes designed for particular situations.

Approach: We have developed *Copa*, an end-to-end congestion control method that achieves these goals. Inspired by work on Network Utility Maximization (NUM) [18] and by machine-generated algorithms, we start with an objective function to optimize. The objective function we use combines a flow's average throughput, λ, and packet delay (minus propagation delay), d: $U = \log \lambda - \delta \log d$. 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.

We show that under certain simplified (but reasonable) modeling assumptions of packet arrivals, the steady-state sending rate (in packets per second) that maximizes U is

$$\lambda = \frac{1}{\delta \cdot d_q}, \quad (1)$$

where d_q is the mean per-packet queuing delay (in seconds), and $1/\delta$ is in units of MTU-sized packets. When every sender transmits at this rate, a unique, socially-acceptable Nash equilibrium is attained.

We use this rate as the *target rate* for a Copa sender. The sender estimates the queuing delay using its RTT observations, moves quickly toward hovering near this target rate. This mechanism also induces a property that the queue is regularly almost flushed, which helps all endpoints get a correct estimate of the queuing delay. Finally, to compete well with buffer-filling competing flows, Copa mimics an AIMD window-update rule when it observes that the bottleneck queues rarely empty.

Results: We have conducted several experiments in emulation, over real-world Internet paths and in simulation comparing Copa to several other methods.

1. As flows enter and leave an emulated network, Copa maintains nearly full link utilization with a median Jain's fairness index of 0.86. The median indices for Cubic, BBR and PCC are 0.81, 0.61 and 0.35 respectively (higher the better).
2. In real-world experiments Copa achieved nearly as much throughput and 2-10× lower queueing delays than Cubic and BBR.
3. In datacenter network simulations, on a web search workload trace drawn from datacenter network [11], Copa achieved a $>5\times$ reduction in flow completion time for short flows over DCTCP. It achieved similar performance for long flows.
4. In experiments on an emulated satellite path, Copa achieved nearly full link utilization with an average queuing delay of only 1 ms. Remy's performance was similar, while PCC achieved similar throughput but with $\approx$700 ms of queuing delay. BBR obtained 50% link utilization with $\approx$100ms queuing delay. Both Cubic and Vegas obtained $< 4\%$ utilization.
5. In an experiment to test RTT-fairness, Copa, Cubic, Cubic over CoDel and Newreno obtained Jain fairness indices of 0.76, 0.12, 0.57 and 0.37 respectively (higher the better). Copa is designed to coexist with TCPs (see section §2.2), but when told that no competing TCPs exist, Copa allocated equal bandwidth to all flows (fairness index $\approx$1).
6. Copa co-exists well with TCP Cubic. On a set of randomly chosen emulated networks where Copa and Cubic flows share a bottleneck, Copa flows benefit and Cubic flows aren't hurt (upto statistically insignificant differences) on average throughput. BBR and PCC obtain higher throughput at the cost of competing Cubic flows.

2 Copa Algorithm

Copa incorporates three ideas: first, a target rate to aim for, which is inversely proportional to the measured queueing delay; second, a window update rule that depends moves the sender toward the target rate; and third, a TCP-competitive strategy to compete well with buffer-filling flows.

2.1 Target Rate and Update Rule

Copa uses a congestion window, cwnd, which upper-bounds the number of in-flight packets. On

every ACK, the sender estimates the current rate λ = cwnd/RTTstanding, where RTTstanding is the smallest RTT observed over a recent time-window, τ. We use τ=srtt/2, where srtt is the current value of the standard smoothed RTT estimate. RTTstanding is the RTT corresponding to a "standing" queue, since it's the minimum observed in a recent time window.

The sender calculates the target rate using Eq. (1), estimating the queueing delay as

$$d_q = \text{RTTstanding} - \text{RTTmin}, \quad (2)$$

where RTTmin is the smallest RTT observed over a long period of time. We use the smaller of 10 seconds and the time since the flow started for this period (the 10-second part is to handle route changes that might alter the minimum RTT of the path).

If the current rate exceeds the target, the sender reduces cwnd; otherwise, it increases cwnd. To avoid packet bursts, the sender paces packets at a rate of 2·cwnd/RTTstanding packets per second. Pacing also makes packet arrivals at the bottleneck queue appear Poisson as the number of flows increases, a useful property that increases the accuracy of our model to derive the target rate (§4). The pacing rate is *double* cwnd/RTTstanding to accommodate imperfections in pacing; if it were exactly cwnd/RTTstanding, then the sender may send slower than desired.

The reason for using the smallest RTT in the recent τ=srtt/2 duration, rather than the latest RTT sample, is for robustness in the face of ACK compression [39] 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. ACK compression can be caused by queuing on the reverse path and by wireless links.

The Copa sender runs the following steps on each ACK arrival:

1. Update the queuing delay d_q using Eq. (2) and srtt using the standard TCP exponentially weighted moving average estimator.
2. Set $\lambda_t = 1/(\delta \cdot d_q)$ according to Eq. (1).
3. If λ = cwnd/RTTstanding $\leq \lambda_t$, then cwnd = cwnd + $v/(\delta \cdot \text{cwnd})$, where v is a "velocity parameter" (defined in the next step). Otherwise, cwnd = cwnd $- v/(\delta \cdot \text{cwnd})$. Over 1 RTT, the change in cwnd is thus $\approx v/\delta$ packets.
4. 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). If the current cwnd is larger, then set *direction* to "up"; if it is smaller, then set *direction* to "down". Now, 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. Since direction may remain the same for 2.5 RTTs in steady state as shown in figure 1, doing otherwise can cause v to be >1 even during steady state. In steady state, we want $v=1$.

When a flow starts, Copa performs slow-start where cwnd doubles once per RTT until λ exceeds λ_t. While the velocity parameter also allows an exponential increase, the constants are smaller. Having an explicit slow-start phase allows Copa to have a larger initial cwnd, like many deployed TCP implementations.

2.2 Competing with Buffer-Filling Schemes

We now modify Copa to compete well with buffer-filling algorithms such as Cubic and NewReno while maintaining its good properties. The problem is that Copa seeks to maintain low queuing delays; without modification, it will lose to buffer-filling schemes.

We propose *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 (Section 3). With even one concurrent long-running buffer-filling flow, the queue will not empty at this periodicity. 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., $d_q < 0.1(\text{RTTmax} - \text{RTTmin})$ where RTTmax is measured over the past four RTTs and RTTmin is our long-term minimum as defined before. Using RTTmax allows Copa to calibrate its notion of "nearly empty" to the amount of short-term RTT variance in the current network.

In competitive mode the sender varies $1/\delta$ according to whatever buffer-filling algorithm one wishes to emulate (e.g., NewReno, Cubic, etc.). In our implementation we perform AIMD on $1/\delta$ based on packet success or loss, but this scheme could respond to other congestion signals. In competitive mode, $\delta \leq 0.5$. When Copa switches from competitive mode to default mode, it resets δ to 0.5.

The queue may be nearly empty even in the presence of a competing buffer-filling flow (e.g., because of a recent packet loss). If that happens, Copa will switch

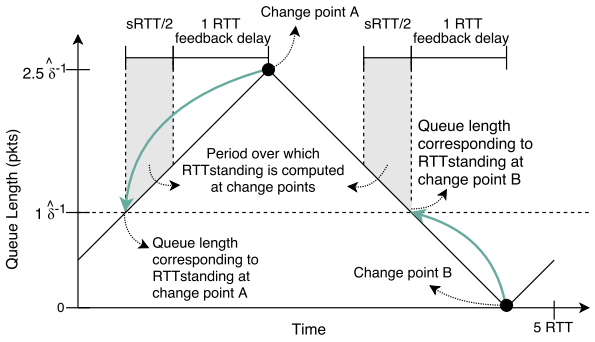

Figure 1: One Copa cycle: Evolution of queue length with time. Copa switches direction at change points A and B when the standing queue length estimated by RTTstanding crosses the threshold of $\hat{\delta}^{-1}$. RTTstanding is the smallest RTT in the last srtt/2 window of ACKs packets (shaded region). Feedback on current actions is delayed by 1 RTT in the network. The slope of the line is $\pm\hat{\delta}$ packets per RTT.

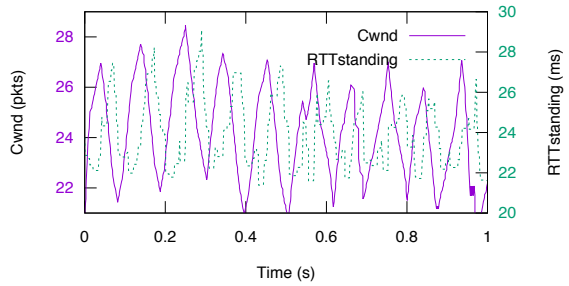

Figure 2: Congestion window and RTT as a function of time for a Copa flow running on a 12 Mbit/s Mahimahi [25] emulated link. As predicted, the period of oscillation is $\approx$5RTT and amplitude is $\approx$5 packets. The emulator's scheduling policies cause irregularities in the RTT measurement, but Copa is immune to such irregularities because the cwnd evolution depends only on comparing RTTstanding to a threshold.

to default mode. Eventually, the buffer will fill again, making Copa switch to competitive mode.

Note that if some Copa flows are operating in competitive mode but no buffer-filling flows are present, perhaps because the decision was erroneous or because the competing flows left the network, Copa flows once again begin to periodically empty the queue. The mode-selection method will detect this condition and switch to default mode.

3 Dynamics of Copa

Figures 1 (schematic view) and 2 (emulated link) show the evolution of Copa's cwnd with time. In steady state, each Copa flow makes small oscillations about the target rate, which also is the equilibrium rate (Section 4). By "equilibrium", we mean the situation when every sender is sending at its target rate. When the propagation delays for flows sharing a bottleneck are similar and comparable to (or larger than) the queuing delay, the small oscillations synchronize to cause the queue length at the bottleneck to oscillate between having 0 and $2.5/\hat{\delta}$ packets every five RTTs. Here, $\hat{\delta} = (\sum_i 1/\delta_i)^{-1}$. The equilibrium queue length is $(0+2.5)\hat{\delta}^{-1}/2 = 1.25/\hat{\delta}$ packets. When each $\delta = 0.5$ (the default value), $1/\hat{\delta} = 2n$, where n is the number of flows.

We prove the above assertions about the steady state using a window analysis for a simplified deterministic (D/D/1) bottleneck queue model. In Section 4 we discuss Markovian (M/M/1 and M/D/1) queues. We assume that the link rate, μ, is constant (or changes slowly compared to the RTT), and that (for simplicity) the feedback delay is constant, RTTmin $\approx$ RTT. This means that the queue length inferred from an ACK at time t is $q(t) = w(t - \text{RTTmin}) - BDP$, where $w(t)$ is congestion window at time t and BDP is the bandwidth-delay product. Under the constant-delay assumption, the sending rate is cwnd/RTT = cwnd/RTTmin.

First consider just one Copa sender. We show that Copa remains in steady state oscillations shown in Figure 1, once it starts those oscillations. In steady state, $v = 1$ (v starts to double only after cwnd changes in the same direction for at least 3 RTTs. In steady state, direction changes once every 2.5 RTT. Hence $v = 1$ in steady state.). When the flow reaches "change point A", its RTTstanding estimate corresponds to minimum in the $\frac{1}{2}$srtt window of latest ACKs. Latest ACKs correspond to packets sent 1 RTT ago. At equilibrium, when the target rate, $\lambda_t = 1/(\delta d_q)$, equals the actual rate, cwnd/RTT, there are $1/\delta$ packets in the queue. When the queue length crosses this threshold of $1/\delta$ packets, the target rate becomes smaller than the current rate. Hence the sender begins to decrease cwnd. By the time the flow reaches "change point B", the queue length has dropped to 0 packets, since cwnd decreases by $1/\delta$ packets per RTT, and it takes 1 RTT for the sender to know that queue length has dropped below target. At "change point B", the rate begins to increase again, continuing the cycle. The resulting mean queue length of the cycle, $1.25/\delta$, is a little higher than $1/\delta$ because RTTstanding takes an extra srtt/2 to reach the threshold at "change point A".

When N senders each with a different δ_i share the bottleneck link, they synchronize with respect to the common delay signal. When they all have the same propagation delay, their target rates cross their actual rates at the same time, irrespective of their δ_i. Hence they increase/decrease their cwnd together, behaving as one sender with $\delta = \hat{\delta} = (\sum_i 1/\delta_i)^{-1}$.

To bootstrap the above steady-state oscillation, target rate should be either above or below the current rate of every sender for at least 1.5 RTT while each $v=1$. Note, other modes of oscillation are possible, for instance when two senders oscillate about the equilibrium rate 180^o out-of-phase, keeping the queue length constant. Nevertheless, small perturbations will cause the target rate to go above/below every sender's current rate, causing the steady-state oscillations described above to commence. So far, we have assumed $v=1$. In practice, we find that the velocity parameter, v, allows the system to quickly reach the target rate. Then it v remains equal to 1 as the senders hover around the target.

To validate our claims empirically, we simulated a dumbbell topology with a 100 Mbit/s bottleneck link, 20 ms propagation delay, and 5 BDP of buffer in ns-2. We introduce flows one by one until 100 flows share the network. We found that the above properties held throughout the simulation. The velocity parameter v remained equal to 1 most of the time, changing only when flow was far from the equilibrium rate. Indeed, these claims hold in most of our experiments, even when jitter is intentionally added.

We have found that this behavior breaks only under two conditions in practice: (1) when the propagation delay is *much smaller* than the queuing delay and (2) when different senders have very different propagation delays, and the delay synchronization weakens. These violations can cause the endpoints to incorrectly think that a competing buffer-filling flow is present(see §2.2). Even in competitive mode, Copa offers several advantages over TCP, including better RTT fairness, better convergence to a fair rate, and loss-resilience.

Median RTTstanding If a flow achieves an steady-state rate of $\approx\lambda$ pkts/s, the median standing queuing delay, RTTstanding − RTTmin, is $1/(\lambda\delta)$. If the median RTTstanding were lower, Copa would increase its rate more often than it decreases, thus increasing λ. Similar reasoning holds if RTTstanding were higher. This means that Copa achieves quantifiably low delay. For instance, if each flow achieves $1.2Mbit/s$ rate in the default mode ($\delta = 0.5$), the median equilibrium queuing delay will be 20 ms. If it achieves 12 Mbit/s, the median equilibrium queuing delay will be 2 ms. In this analysis, we neglect the variation in λ during steady state oscillations since it is small.

Alternate approaches to reaching equilibrium. A different approach would be to *directly* set the current sending rate to the target rate of $1/\delta d_q$. We experimented with and analyzed this approach, but found that the system converges only under certain conditions. We proved that the system converges to a constant rate when $C\cdot\sum_i 1/\delta_i <$ (bandwidth delay product), where $C\approx 0.8$ is a dimensionless constant. With ns-2 simulations, we found this condition to be both necessary and sufficient for convergence. Otherwise it oscillates. These oscillations can lead to severe underutilization of the network and it is non-trivial to ensure that we always operate at the condition where convergence is guaranteed.

Moreover, convergence to a constant rate and non-zero queuing delay is not ideal for a delay-based congestion controller. If the queue never empties, flows that arrive later will over-estimate their minimum RTT and hence underestimate their queuing delay. This leads to significant unfairness. Thus, we need a scheme that approaches the equilibrium incrementally and makes small oscillations about the equilibrium to regularly drain the queues.

A natural alternative candidate to Copa's method is additive-increase/multiplicative-decrease (AIMD) when the rate is below or above the target. However, Copa's objective function seeks to keep the queue length small. If a multiplicative decrease is performed at this point, severe under-utilization occurs. Similarly, a multiplicative increase near the equilibrium point will cause a large queue length.

AIAD meets many of our requirements. It converges to the equilibrium and makes small oscillations about it such that the queue is periodically emptied, while maintaining a high link utilization (§5.1). However, if the bandwidth-delay product (BDP) is large, AIAD can take a long time to reach equilibrium. Hence we introduce a velocity parameter §2.1 that moves the rate exponentially fast toward the equilibrium point, after which it uses AIAD.

4 Justification of the Copa Target Rate

This section explains the rationale for the target rate used in Copa. We model packet arrivals at a bottleneck not as deterministic arrivals as in the previous section, but as Poisson arrivals. This is a simplifying assumption, but one that is more realistic than deterministic arrivals when there are multiple flows. The key property of random packet arrivals (such as with a Poisson distribution) is that queues build up even when the bottleneck link is not fully utilized.

In general traffic may be burstier than predicted by Poisson arrivals [28] because flows and packet transmissions can be correlated with each other. In this case, Copa over-estimates network load and responds by implicitly valuing delay more. This behavior is reasonable as increased risk of higher delay is being met by more caution. Ultimately, our validation of the Copa algorithm is through experiments, but the modeling assumption provides a sound basis for setting a good target rate.

4.1 Objective Function and Nash Equilibrium

Consider the objective function for sender (flow) i combining both throughput and delay:

$$U_i = \log \lambda_i - \delta_i \log d_s, \tag{3}$$

where $d_s = d_q + 1/\mu$ is the "switch delay" (total minus propagation delay). The use of switch delay is for technical ease; it is nearly equal to the queuing delay.

Suppose each sender attempts to maximize its own objective function. In this model, the system will be at a Nash equilibrium when no sender can increase its objective function by unilaterally changing its rate. The Nash equilibrium is the n-tuple of sending rates $(\lambda_1,...,\lambda_n)$ satisfying

$$U_i(\lambda_1,...,\lambda_i,...,\lambda_n) > U_i(\lambda_1,...,\lambda_{i-1},x,\lambda_{i+1},...,\lambda_n) \tag{4}$$

for all senders i and any non-negative x.

We assume a first-order approximation of Markovian packet arrivals. The service process of the bottleneck may be random (due to cross traffic, or time-varying link rates), or deterministic (fixed-rate links, no cross traffic). As a reasonable first-order model of the random service process at the bottleneck link, we assume a Markovian service distribution and use that model to develop the Copa update rule. Assuming a deterministic service process gives similar results, offset by a factor of 2. In principle, senders could send their data not at a certain mean rate but in Markovian fashion, which would make our modeling assumption match practice. In practice, we don't, because: (1) there is natural jitter in transmissions from endpoints anyway, (2) deliberate jitter unnecessarily increases delay when there are a small number of senders and, (3) Copa's behavior is not sensitive to the assumption.

We prove the following proposition about the existence of a Nash equilibrium for Markovian packet transmissions. We then use the properties of this equilibrium to derive the Copa target rate of Eq. (1). The reason we are interested in the equilibrium property is that the rate-update rule is intended to optimize each sender's utility independently; we derive it directly from this theoretical rate at the Nash equilibrium. It is important to note that this model is being used not because it is precise, but because it is a simple and tractable approximation of reality. Our goal is to derive a principled target rate that arises as a stable point of the model, and use that to guide the rate update rule.

Lemma 1. *Consider a network with n flows, with flow i sending packets with rate λ_i such that the arrival at the bottleneck queue is Markovian. Then, if flow i has the objective function defined by Eq. (3), and the bottleneck is an M/M/1 queue, a unique Nash equilibrium exists. Further, at this equilibrium, for every sender i,*

$$\lambda_i = \frac{\mu}{\delta_i(\hat{\delta}^{-1}+1)} \tag{5}$$

where $\hat{\delta} = (\sum 1/\delta_i)^{-1}$.

Proof. Denote the total arrival rate in the queue, $\sum_j \lambda_j$, by λ. For an M/M/1 queue, the sum of the average wait time in the queue and the link is $\frac{1}{\mu-\lambda}$. Substituting this expression into Eq. (3) and separating out the λ_i term, we get

$$U_i = \log \lambda_i + \delta_i \log(\mu - \lambda_i - \sum_{j\neq i} \lambda_j). \tag{6}$$

Setting the partial derivative $\frac{\partial U_i}{\partial \lambda_i}$ to 0 for each i yields

$$\delta_i \lambda_i + \sum_j \lambda_j = \mu$$

The second derivative, $-1/\lambda_i^2 - \delta_i/(\mu - \lambda)^2$, is negative.

Hence Eq. (4) is satisfied if, and only if, $\forall i$, $\frac{\partial U_i}{\partial \lambda_i} = 0$. We obtain the following set of n equations, one for each sender i:

$$\lambda_i(1+\delta_i) + \sum_{j\neq i} \lambda_j = \mu.$$

The unique solution to this family of linear equations is

$$\lambda_i = \frac{\mu}{\delta_i(\hat{\delta}^{-1}+1)},$$

which is the desired equilibrium rate of sender i. □

When the service process is assumed to be deterministic, we can model the network as an M/D/1 queue. The expected wait time in the queue is $1/(2(\mu-\lambda)) - \mu/2 \approx 1/2(\mu-\lambda)$. An analysis similar to above gives the equilibrium rate of sender i to be $\lambda_i = 2\mu/(\delta_i(2\hat{\delta}^{-1}+1))$, which is the same as the M/M/1 case when each δ_i is halved. Since there is less uncertainty, senders can achieve higher rates for the same delay.

4.2 The Copa Update Rule Follows from the Equilibrium Rate

At equilibrium, the inter-send time between packets is

$$\tau_i = \frac{1}{\lambda_i} = \frac{\delta_i(\hat{\delta}^{-1}+1)}{\mu}.$$

Each sender does not, however, need to know how many other senders there are, nor what their δ_i preferences are, thanks to the aggregate behavior of Markovian arrivals. The term inside the parentheses in the equation above is a proxy for the "effective" number of other senders, or equivalently the network load, and can be calculated differently.

As noted earlier, the average switch delay for an M/M/1 queue is $d_s = \frac{1}{\mu-\lambda}$. Substituting Eq, (8) for λ in this equation, we find that, at equilibrium,

$$\tau_i = \delta_i \cdot d_s = \delta_i(d_q + 1/\mu), \quad (7)$$

where d_s is the switch delay (as defined earlier) and d_q is the average queuing delay in the network.

This calculation is the basis and inspiration for the target rate. The *does not* model the dynamics of Copa, where sender rates change with time. The purpose of this analysis is to determine a good target rate for senders to aim for. Nevertheless, using steady state formulae for expected queue delay is acceptable since the rates change slowly in steady state.

4.3 Properties of the Equilibrium

We now make some remarks about this equilibrium. First, by adding Eq. (5) over all i, we find that the resulting aggregate rate of all senders is

$$\lambda = \sum \lambda_j = \mu/(1+\hat{\delta}) \quad (8)$$

This also means that the equilibrium queuing delay is $1+1/\hat{\delta}$. If $\delta_i = 0.5$, the number of enqueued packets with n flows is $2n+1$.

Second, it is interesting to interpret Eqs. (5) and (8) in the important special case when the δ_is are all the same δ. Then, $\lambda_i = \mu/(\delta n)$, which is equivalent to dividing the capacity between n senders and δ (which may be non-integral) "pseudo-senders". δ is the "gap" from fully loading the bottleneck link to allow the average packet delay to not blow up to ∞. The portion of capacity allocated to "pseudo-senders" is unused and determines the average queue length which the senders can adjust by choosing any $\delta \in (0,\infty)$. The aggregate rate in this case is $n \cdot \lambda_i = \frac{n\mu}{\delta+n}$. When δ_is are unequal, bandwidth is allocated in inverse proportion to δ_i. The Copa rate update rules are such that a sender with constant parameter δ is equivalent to k senders with a constant parameter $k\delta$ in steady state.

Third, we recommend a default value of $\delta_i = 0.5$. While we want low delay, we also want high throughput; i.e., we want the largest δ that also achieves high throughput. A value of 1 causes one packet in the queue on average at equilibrium (i.e., when the sender transmits at the target equilibrium rate). While acceptable in theory, jitter causes packets to be imperfectly paced in practice, causing frequently empty queues and wasted transmission slots when a only single flow occupies a bottleneck, a common occurrence in our experience. Hence we choose $\delta = 1/2$, providing headroom for packet pacing. Note that, as per the above equation modeled on an M/M/1 queue, the link would be severely underutilized when there are a small number (≤ 5) of senders. But with very few senders, arrivals at the queue aren't Poisson and stochastic variations don't cause the queue length to rise. Hence link utilization is nearly 100% before queues grow as demonstrated in §5.1.

Fourth, the definition of the equilibrium point is consistent with our update rule in the sense that every sender's transmission rate equals their target rate if (and only if) the system is at the Nash equilibrium. This analysis presents a mechanism to determine the behavior of a cooperating sender: every sender observes a common delay d_s and calculates a common δd_s (if all senders have the same δ) or its $\delta_i d_s$. Those transmitting faster than the reciprocal of this value must reduce their rate and those transmitting slower must increase it. If every sender behaves thus, they will all benefit.

5 Evaluation

To evaluate Copa and compare it with other congestion-control protocols, we use a user-space implementation and ns-2 simulations. We run the user-space implementation over both emulated and real links.

Implementations: We compare the performance of our user-space implementation of Copa with Linux kernel implementations of TCP Cubic, Vegas, Reno, and BBR [5], and user-space implementations of Remy, PCC [6], Vivace [7], Sprout [36], and Verus [38]. We used the developers' implementations for PCC and Sprout. For Remy, we developed a user-space implementation and verified that its results matched the Remy simulator. There are many available RemyCCs and whenever we found a RemyCC that was appropriate for that network, we report its results. We use the Linux qdisc to create emulated links. Our PCC results are for the default loss-based objective function. Pantheon [37], an independent test-bed for congestion control, uses the delay-based objective function for PCC.

ns-2 simulations: We compare Copa with Cubic [13], NewReno [15], and Vegas [4], which are end-to-end protocols, and with Cubic-over-CoDel [26] and DCTCP [1], which use in-network mechanisms.

5.1 Dynamic Behavior over Emulated Links

To understand how Copa behaves as flows arrive and leave, we set up a 100 Mbit/s link with 20 ms RTT and 1 BDP buffer using Linux qdiscs. One flow arrives every second for the first ten seconds, and one leaves every second for the next ten seconds. The mean $\pm$ standard deviation of the bandwidths obtained by the flows at each time slot are shown in Figure 3. A CDF of the Jain fairness index in various timeslots is shown in Figure 4.

Both Copa and Cubic track the ideal rate allocation. Figure 4 shows that Copa has the highest median fairness index, with Cubic close behind. BBR and PCC respond much more slowly to changing network

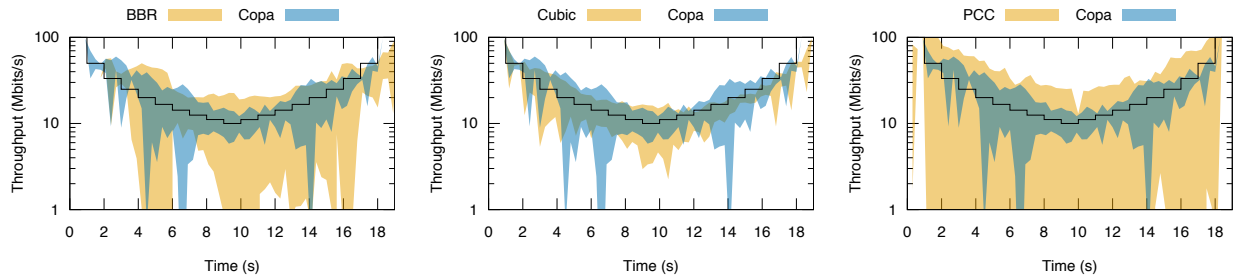

Figure 3: Mean ± std. deviation of throughput of 10 flows as they enter and leave the network once a second. The black line indicates the ideal allocation. Graphs for BBR, Cubic and PCC are shown alongside Copa in each figure for comparison. Copa and Cubic flows follow the ideal allocation closely.

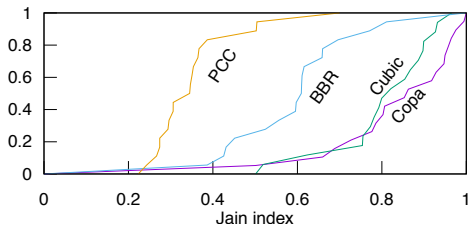

Figure 4: A CDF of the Jain indices obtained at various timeslots for the dynamic behavior experiment (§5.1)

conditions and fail to properly allocate bandwidth. In experiments where the network changed more slowly, BBR and PCC eventually succeeded in converging to the fair allocation, but this took tens of seconds.

This experiment shows Copa's ability to quickly adapt to changing environments. Copa's mode switcher correctly functioned most of the time, detecting that no buffer-filling algorithms were active in this period. Much of the noise and unfairness observed in Copa in this experiment was due erroneous switches to competitive mode for a few RTTs. This happens because when flows arrive or depart, they disturb Copa's steady-state operation. Hence it is possible that for a few RTTs the queue is never empty and Copa flows can switch from default to competitive mode. In this experiment, there were a few RTTs during which several flows switched to competitive mode, and their δ decreased. However, queues empty every five RTTs in this mode as well if no competing buffer-filling flow is present. This property enabled Copa to correctly revert to default mode after a few RTTs.

5.2 Real-World Evaluation

To understand how Copa performs over wide-area Internet paths with real cross traffic and packet schedulers, we submitted our user-space implementation of Copa to Pantheon [37] (`http://pantheon.stanford.edu`), a system developed to evaluate congestion control schemes. During our experiments, Pantheon had nodes in six countries. It creates flows using each congestion control scheme between a node and an AWS server nearest it, and measures the throughput and delay. We separate the set of experiments into two categories, depending on how the node connects to the Internet (Ethernet or cellular).

To obtain an aggregate view of performance across the dozens of experiments, we plot the average normalized throughput and average queuing delay. Throughput is normalized relative to the flow that obtained the highest throughput among all runs in an experiment to obtain a number between 0 and 1. Pantheon reports the one-way delay for every packet in publicly-accessible logs calculated with NTP-synchronized clocks at the two end hosts. To avoid being confounded by the systematic additive delay inherent in NTP, we report the queuing delay, calculated as the difference between the delay and the minimum delay seen for that flow. Each experiment lasts 30 seconds. Half of them have one flow. The other half have three flows starting at 0, 10, and 20 seconds from the start of the experiment.

Copa's performance is consistent across different types of networks. It achieves significantly lower queueing delays than most other schemes, with only a small throughput reduction. Copa's low delay, loss insensitivity, RTT fairness, resistance to buffer-bloat, and fast convergence enable resilience in a wide variety of network settings. Vivace LTE and Vivace latency achieved excessive delays in a link between AWS São Paulo and a node in Columbia, sometimes over 10 seconds. When all runs with > 2000 ms are removed for Vivace latency and LTE, they obtain average queuing delays of 156 ms and 240 ms respectively, still significantly higher than Copa. The Remy method used was trained for a $100\times$ range of link rates. PCC uses its delay-based objective function.

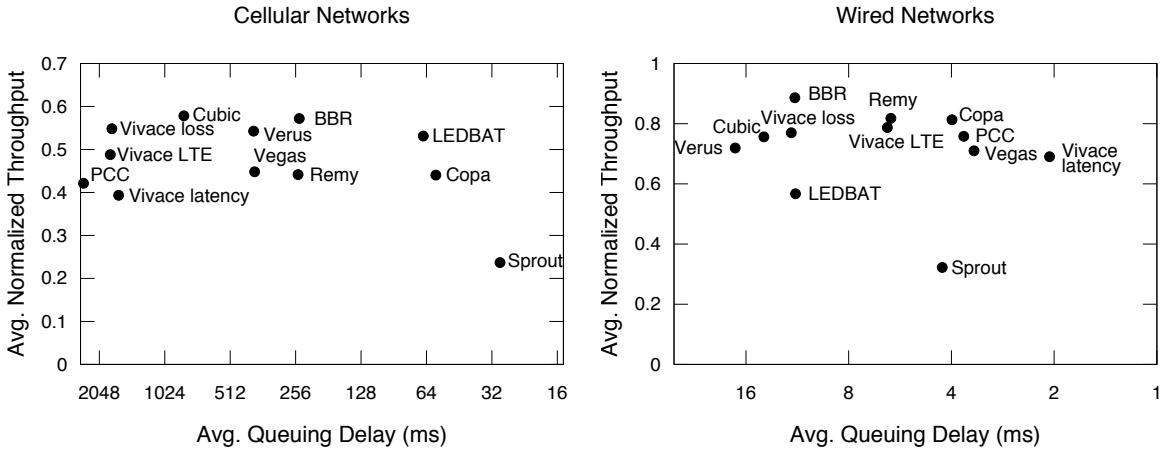

Figure 5: Real-world experiments on Pantheon paths: Average normalized throughput vs. queuing delay achieved by various congestion control algorithms under two different types of Internet connections. Each type is averaged over several runs over 6 Internet paths. Note the very different axis ranges in 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 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× lower than BBR and Cubic, with only a modest mean throughput reduction.

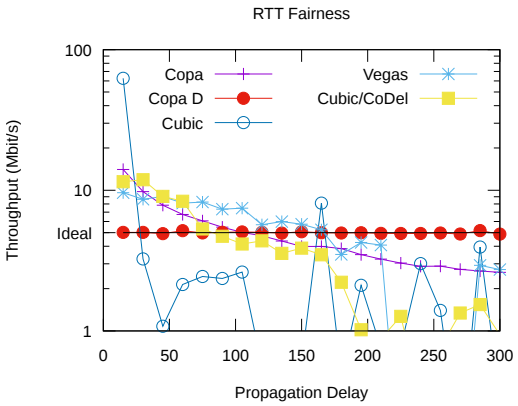

Figure 6: RTT-fairness of various schemes. Throughput of 20 long-running flows sharing a 100 Mbit/s bottleneck link versus their respective propagation delays. "Copa D" is Copa in the default mode without mode switching. "Copa" is the full algorithm. "Ideal" shows the fair allocation, which "Copa D" matches. Notice the log scale on the y-axis. Schemes other than name allocate little bandwidth to flows with large RTTs.

5.3 RTT-fairness

Flows sharing the same bottleneck link often have different propagation delays. Ideally, they should get identical throughput, but many algorithms exhibit significant RTT unfairness, disadvantaging flows with larger RTTs. To evaluate the RTT fairness of various algorithms, we set up 20 long-running flows in ns-2 with propagation delays evenly spaced between 15 ms and 300 ms. The link has a bandwidth of 100 Mbit/s and 1 BDP of buffer (calculated with 300 ms delay). The experiment runs for 100 seconds. We plot the throughput obtained by each of the flows in Figure 6.

Copa's property that the queue is nearly empty once in every five RTTs is violated when such a diversity of propagation delays is present. Hence Copa's mode switching algorithm erroneously shifts to competitive mode, causing Copa with mode switching (labeled "Copa" in the figure) to inherit AIMD's RTT unfriendliness. However, because the AIMD is on $1/\delta$ while the underlying delay-sensitive algorithm robustly grabs or relinquishes bandwidth to make the allocation proportional to $1/\delta$, Copa's RTT-unfriendliness is much milder than in the other schemes.

We also run Copa after turning off the mode-switching and running it in the default mode ($\delta = 0.5$), denoted as "Copa D" in the figure. Because the senders share a common queuing delay and a common target rate, under identical conditions, they will make identical decisions to increase/decrease their rate, but with a time shift. This approach removes any RTT bias, as shown by "Copa D".

In principle, Cubic has a window evolution that is RTT-independent, but in practice it exhibits significant RTT-unfairness because low-RTT Cubic senders are slow to relinquish bandwidth. The presence of the CoDel AQM improves the situation, but significant unfairness remains. Vegas is unfair because several

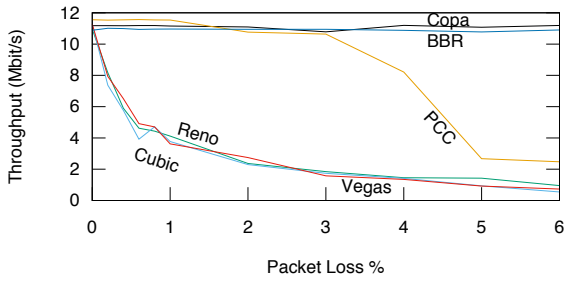

Figure 7: Performance of various schemes in the presence of stochastic packet loss over a 12 Mbit/s link with a 50 ms RTT.

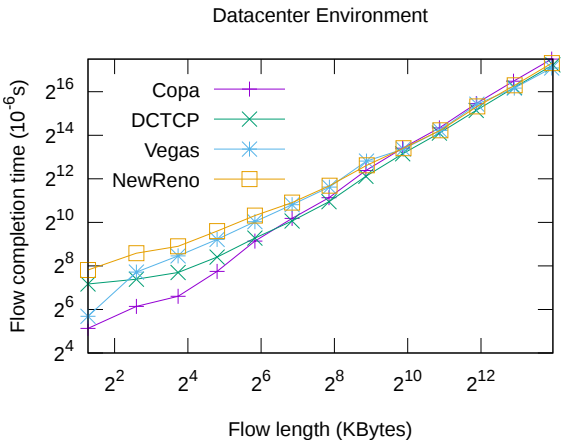

Figure 8: Flow completion times achieved by various schemes in a datacenter environment. Note the log scale.

flows have incorrect base RTT estimates as the queue rarely drains. Schemes other than Copa allocate nearly no bandwidth to long RTT flows (note the log scale), a problem that Copa solves.

5.4 Robustness to Packet Loss

To meet the expectations of loss-based congestion control schemes, lower layers of modern networks attempt to hide packet losses by implementing extensive reliability mechanisms. These often lead to excessively high and variable link-layer delays, as in many cellular networks. Loss is also sometimes blamed for the poor performance of congestion control schemes across trans-continental links (we have confirmed this with measurements, e.g., between AWS in Europe and non-AWS nodes in the US). Ideally, a 5% non-congestive packet loss rate should decrease the throughput by 5%, not by 5×. Since TCP requires smaller loss rates for larger window sizes, loss resilience becomes more important as network bandwidth rises.

Copa in default mode does not use loss as a congestion signal and lost packets only impact Copa to the extent that they occupy wasted transmission slots in the congestion window. In the presence of high packet loss, Copa's mode switcher would switch to default mode as any competing traditional TCPs will back off. Hence Copa should be largely insensitive to stochastic loss, while still performing sound congestion control.

To test this hypothesis, we set up an emulated link with a rate of 12 Mbit/s bandwidth and an RTT of 50 ms. We vary the stochastic packet loss rate and plot the throughput obtained by various algorithms. Each flow runs for 60 seconds.

Figure 7 shows the results. Copa and BBR remain insensitive to loss throughout the range, validating our hypothesis. As predicted [22], NewReno, Cubic, and Vegas decline in throughput with increasing loss rate. PCC ignores loss rates up to $\approx$5%, and so maintains throughput until then, before falling off sharply as determined by its sigmoid loss function.

5.5 Simulated Datacenter Network

To test how widely beneficial the ideas in Copa might be, we consider datacenter networks, which have radically different properties than wide-area networks. Many congestion-control algorithms for datacenters exploit the fact that one entity owns and controls the entire network, which makes it easier to incorporate in-network support [1, 3, 24, 29, 12]. DCTCP [1] and Timely [23], for example, aim to detect and respond to congestion before it builds up. DCTCP uses routers to mark ECN in packet headers when queue length exceeds a pre-set threshold. Timely demonstrates that modern commodity hardware is precise enough to accurately measure RTT and pace packets. Hence it uses delay as a fine-grained signal for monitoring congestion. Copa is similar in its tendency to move toward a target rate in response to congestion.

Exploiting the datacenter's controlled environment, we make three small changes to the algorithm: (1) the propagation delay is externally provided, (2) since it is not necessary to compete with TCPs, we disable the mode switching and always operate at the default mode with $\delta = 0.5$ and, (3) since network jitter is absent, we use the latest RTT instead of RTTstanding, which also enables faster convergence. For computing v, the congestion window is considered to change in a given direction only if $> 2/3$ of ACKs cause motion in that direction.

We simulate 32 senders connected to a 40 Gbit/s bottleneck link via 10 Gbit/s links. The routers have 600 Kbytes of buffer and each flow has a propagation delay of 12 μs. We use an on-off workload with flow lengths drawn from a web-search workload in the

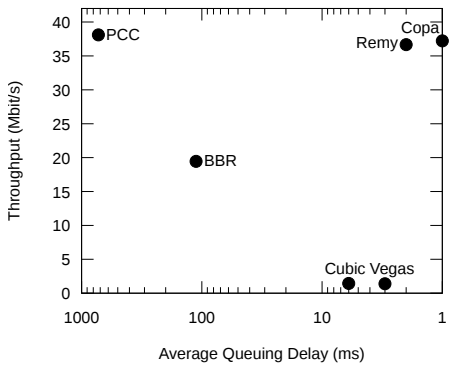

Figure 9: Throughput vs. delay plot for a satellite link. Notice that algorithms that are not very loss sensitive (including PCC, which ignores small loss rates) all do well on throughput, but the delay-sensitive ones get substantially lower delay as well. Note the log-scale.

datacenter [1]. Off times are exponentially distributed with mean 200 ms. We compare Copa to DCTCP, Vegas, and NewReno.

The average flow completion times (FCT) are plotted against the length of the flow in Figure 8, with the y-axis shown on a log-scale. Because of its tendency to maintain short queues and robustly converge to equilibrium, Copa offers significant reduction in flow-completion time (FCT) for short flows. The FCT of Copa is between a factor of 3 and 16 better for small flows under 64 kbytes compared to DCTCP. For longer flows, the benefits are modest, and in many cases other schemes perform a little better in the datacenter setting. This result suggests that Copa is a good solution for datacenter network workloads involving short flows.

We also implemented TIMELY [23], but it did not perform well in this setting (over 7 times worse than Copa on average), possibly because TIMELY is targeted at getting high throughput and low delay for long flows. TIMELY requires several parameters to be set; we communicated with the developers and used their recommended parameters, but the difference between our workload and their RDMA experiments could explain the discrepancies; because we are not certain, we do not report those results in the graph.

5.6 Emulated Satellite Links

We evaluate Copa on an emulated satellite link using measurements from the WINDS satellite system [27], replicating an experiment from the PCC paper [6]. The link has a 42 Mbit/s capacity, 800 ms RTT, 1 BDP of buffer and 0.74% stochastic loss rate, on which we run 2 concurrent senders for 100 seconds. This link is challenging because it has a high bandwidth-delay product and some stochastic loss.

Figure 9 shows the throughput v. delay plot for BBR, PCC, Remy, Cubic, Vegas, and Copa. Here we use a RemyCC trained for a RTT range of 30-280 ms for 2 senders with exponential on-off traffic of 1 second, each over a link speed of 33 Mbit/s, which, surprisingly, worked well in this case and was the best performer among the ones available in the Remy repository.

PCC obtained high throughput, but at the cost of high delay as it tends to fill the buffer. BBR ignores loss, but still underutilized the link as its rate oscillated wildly between 0 and over 42 Mbit/s due to the high BDP. These oscillations also causing high delays. Copa is insensitive to loss and can scale to large BDPs due to its exponential rate update. Both Cubic and Vegas are sensitive to loss and hence lose throughput.

5.7 Co-existence with Buffer-Filling Schemes

A major concern is whether current TCP algorithms will simply overwhelm the delay-sensitivity embedded in Copa. We ask: (1) how does Copa affects existing TCP flows?, and (2) do Copa flows get their fair share of bandwidth when competing with TCP (i.e., how well does mode-switching work)?

We experiment on several emulated networks. We randomly sample throughput between 1 and 50 Mbit/s, RTT between 2 and 100 ms, buffer size between 0.5 and 5 BDP, and ran 1-4 Cubic senders and 1-4 senders of the congestion control algorithm being evaluated. The flows are run concurrently for 10 seconds. We report the average of the ratio of the throughput achieved by each flow to its ideal fair share for both the algorithm being tested and Cubic. To set a baseline for variations within Cubic, we also report numbers for Cubic, treating one set of Cubic flows as "different" from another.

Figure 10 shows the results. Even when competing with other Cubic flows, Cubic is unable to fully utilize the network. Copa takes this unused capacity to achieve greater throughput without hurting Cubic flows. In fact, Cubic flows competing with Copa get a higher throughput than when competing with other Cubic flows (by a statistically insignificant margin). Currently Copa in competitive mode performs AIMD on $1/\delta$. Modifying this to more closely match Cubic's behavior will help reduce the standard deviation.

PCC gets a much higher share of throughput because its loss-based objective function ignores losses until about 5% and optimizes throughput. BBR gets higher throughput while significantly hurting competing Cubic flows.

6 Related Work

Delay-based schemes like CARD [17], DUAL [33], and Vegas [4] were viewed as ineffective by much of the community for several years, but underwent a revival in the 2000s with FAST [34] and especially Microsoft's

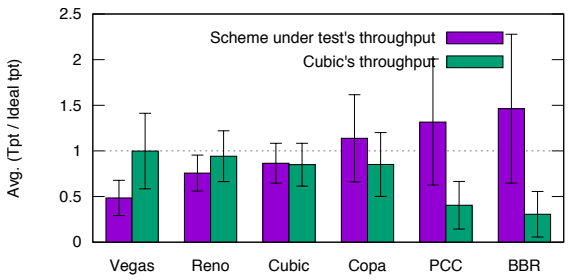

Figure 10: Throughput of different schemes versus Cubic shown by plotting the mean and standard deviation of the ratio of each flow's throughput to the ideal fair rate. The mean is over several runs of randomly sampled networks. The left and right bars show the value for the scheme being tested and Cubic respectively. Copa is much fairer than BBR and PCC to Cubic. It also uses bandwidth that Cubic does not utilize to get higher throughput without hurting Cubic.

Compound TCP [32]. Recently, delay-based control has been used in datacenters by DX [19] and TIMELY [23]. Vegas and FAST share some equilibrium properties with Copa in the sense that they all seek to maintain queue length in proportion to the number of flows. Vegas seeks to maintain between 3 and 6 packets per flow in the queue, and doesn't change its rate if this target is met. Copa's tendency to always change its rate ensures that the queue is periodically empty. This approach has two advantages: (1) every sender gets the correct estimate of minimum RTT, which helps ensure fairness and, (2) Copa can quickly detect the presence of a competing buffer-filling flow and change its aggressiveness accordingly. Further, Copa adapts its rate exponentially allowing it to scale to large-BDP networks. Vegas and FAST increase their rate as a linear function of time.

Network utility maximization (NUM) [18] approaches turn utility maximization problems into rate allocations and vice versa. FAST [34] derives its equilibrium properties from utility maximization to propose an end-to-end congestion control mechanism. Other schemes [24, 14, 24] use the NUM framework to develop with algorithms that use in-network support.

BBR [5] uses bandwidth estimation to operate near the optimal point of full bandwidth utilization and low delay. Although very different from Copa in mechanism, BBR shares some desirable properties with Copa, such as loss insensitivity, better RTT fairness, and resilience to bufferbloat. Experiments §5.2 show that Copa achieves significantly lower delay and slightly less throughput than BBR. There are three reasons for this. First, the default choice of $\delta = 0.5$, intended for interactive applications, encourages Copa to trade a little throughput for a significant reduction in delay. Applications can choose a smaller value of δ to get more throughput, such as $\delta = 0.5/6$, emulating 6 ordinary Copa flows, analogous to how some applications open 6 TCP connections today. Second, BBR tries to be TCP-compatible within its one mechanism. This forced BBR's designers to choose more aggressive parameters, causing longer queues even when competing TCP flows are not present [5]. Copa's use of two different modes with explicit switching allowed us to choose more conservative parameters in the absence of competing flows. Third, both BBR and Copa seek to empty their queues periodically to correctly estimate the propagation delay. BBR uses a separate mechanism with a 10-second cycle, while Copa drains once every 5 RTTs within its AIAD mechanism. As shown in our evaluation in §5.1 and §5.5, Copa is able to adapt more rapidly to changing network conditions. It is also able to handle networks with large-BDP paths better than BBR (§5.6).

7 Conclusion

We described the design and evaluation of Copa, a practical delay-based congestion control algorithm for the Internet. The idea is to increase or decrease the congestion window depending on whether the current rate is lower or higher than a well-defined target rate, $1/(\delta d_q)$, where d_q is the (measured) queueing delay. We showed how this target rate optimizes a natural function of throughput and delay. Copa uses a simple update rule to adjust the congestion window in the direction of the target rate, converging quickly to the correct fair rates even in the face of significant flow churn.

These two ideas enable Copa flows to maintain high utilization with low queuing delay (on average, $1.25/\delta$ packets per flow in the queue). However, when the bottleneck is shared with buffer-filling flows like Cubic or NewReno, Copa, like other delay-sensitive schemes, has low throughput. To combat this problem, a Copa sender detects the presence of buffer-fillers by observing the delay evolution, and then responds with AIMD on the δ parameter to compete well with these schemes.

Acknowledgments

We thank Greg Lauer, Steve Zabele, and Mark Keaton for implementing Copa on BBN's DARPA-funded IRON project and for sharing experimental results, which helped improve Copa. We are grateful to Karthik Gopalakrishnan and Hamsa Balakrishnan for analyzing an initial version of Copa. We thank the anonymous reviewers and our shepherd Dongsu Han for many useful comments. Finally, we would like to thank members of the NMS group at CSAIL, MIT for many interesting and useful discussions. Our work was

funded in part by DARPA's EdgeCT program (award 024890-00001) and NSF grant 1407470.

References

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

[2] M. Alizadeh, A. Kabbani, T. Edsall, B. Prabhakar, A. Vahdat, and M. Yasuda. Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center. In *NSDI*, 2012.

[3] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal Near-Optimal Datacenter Transport. In *SIGCOMM*, 2013.

[4] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *SIGCOMM*, 1994.

[5] N. Cardwell, Y. Cheng, C. S. Gunn, S. H. Yeganeh, and V. Jacobson. Bbr: Congestion-based congestion control. *Queue*, 14(5):50, 2016.

[6] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira. PCC: Re-architecting Congestion Control for Consistent High Performance. In *NSDI*, 2015.

[7] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira. Vivace: Online-learning congestion control. In *15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18)*. USENIX Association, 2018.

[8] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: Vehicular Content Delivery using WiFi. In *MobiCom*, 2008.

[9] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan. Reducing Web Latency: The Virtue of Gentle Aggression. In *SIGCOMM*, 2013.

[10] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: Rate limiting youtube video streaming. In *USENIX ATC*, 2012.

[11] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta. VL2: A Scalable and Flexible Data Center Network. In *SIGCOMM*, 2009.

[12] M. P. Grosvenor, M. Schwarzkopf, I. Gog, R. N. Watson, A. W. Moore, S. Hand, and J. Crowcroft. Queues don't matter when you can jump them! In *NSDI*, pages 1–14, 2015.

[13] S. Ha, I. Rhee, and L. Xu. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *ACM SIGOPS Operating System Review*, 42(5):64–74, July 2008.

[14] D. Han, R. Grandl, A. Akella, and S. Seshan. Fcp: a flexible transport framework for accommodating diversity. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 135–146. ACM, 2013.

[15] J. C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *SIGCOMM*, 1996.

[16] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM*, 1988.

[17] R. Jain. A Delay-based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks. In *SIGCOMM*, 1989.

[18] F. P. Kelly, A. Maulloo, and D. Tan. Rate Control in Communication Networks: Shadow Prices, Proportional Fairness and Stability. *Journal of the Operational Research Society*, 49:237–252, 1998.

[19] C. Lee, C. Park, K. Jang, S. Moon, and D. Han. Accurate Latency-based Congestion Feedback for Datacenters. In *USENIX ATC*, 2015.

[20] Z. Li, X. Zhu, J. Gahm, R. Pan, H. Hu, A. Begen, and D. Oran. Probe and adapt: Rate adaptation for HTTP video streaming at scale. *IEEE JSAC*, 32(4):719–733, 2014.

[21] R. Mahajan, J. Padhye, S. Agarwal, and B. Zill. High Performance Vehicular Connectivity with Opportunistic Erasure Coding. In *USENIX ATC*, 2012.

[22] M. Mathis, J. Semke, J. Mahdavi, and T. Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *ACM SIGCOMM Computer Communication Review*, 27(3):67–82, 1997.

[23] R. Mittal, N. Dukkipati, E. Blem, H. Wassel, M. Ghobadi, A. Vahdat, Y. Wang, D. Wetherall, D. Zats, et al. TIMELY: RTT-based Congestion Control for the Datacenter. In *SIGCOMM*, 2015.

[24] K. Nagaraj, D. Bharadia, H. Mao, S. Chinchali, M. Alizadeh, and S. Katti. Numfabric: Fast and flexible bandwidth allocation in datacenters. In *Proceedings of the 2016 conference on ACM SIGCOMM 2016 Conference*, pages 188–201. ACM, 2016.

[25] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan. Mahimahi: Accurate Record-and-Replay for HTTP. In *USENIX ATC*, 2015.

[26] K. Nichols and V. Jacobson. Controlling Queue Delay. *ACM Queue*, 10(5), May 2012.

[27] H. Obata, K. Tamehiro, and K. Ishida. Experimental evaluation of TCP-STAR for satellite Internet over WINDS. In *International Symposium on Autonomous Decentralized Systems (ISADS)*, 2011.

[28] V. Paxson and S. Floyd. Wide area traffic: the failure of poisson modeling. *IEEE/ACM Transactions on Networking (ToN)*, 3(3):226–244, 1995.

[29] J. Perry, A. Ousterhout, H. Balakrishnan, D. Shah, and H. Fugal. Fastpass: A Centralized "Zero-Queue" Datacenter Network. In *SIGCOMM*, 2014.

[30] A. Sivaraman, K. Winstein, P. Thaker, and H. Balakrishnan. An Experimental Study of the Learnability of Congestion Control. In *SIGCOMM*, 2014.

[31] M. Sridharan, K. Tan, D. Bansal, and D. Thaler. Compound TCP: A New TCP congestion control for high-speed and long distance networks. Technical report, Internet-draft draft-sridharan-tcpm-ctcp-02, 2008.

[32] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A Compound TCP Approach for High-speed and Long Distance Networks. In *INFOCOM*, 2006.

[33] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). In *SIGCOMM*, 1991.

[34] D. Wei, C. Jin, S. Low, and S. Hegde. FAST TCP: Motivation, Architecture, Algorithms, Performance. *IEEE/ACM Trans. on Networking*, 14(6):1246–1259, 2006.

[35] K. Winstein and H. Balakrishnan. TCP ex Machina: Computer-Generated Congestion Control. In *SIGCOMM*, 2013.

[36] K. Winstein, A. Sivaraman, and H. Balakrishnan. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *NSDI*, 2013.

[37] F. Y. Yan, J. Ma, G. Hill, D. Raghavan, R. S. Wahby, P. Levis, and K. Winstein. Pantheon: the training ground for internet congestion-control research. `http://pantheon.stanford.edu/`.

[38] Y. Zaki, T. Pötsch, J. Chen, L. Subramanian, and C. Görg. Adaptive congestion control for unpredictable cellular networks. In *SIGCOMM*, 2015.

[39] L. Zhang, S. Shenker, and D. D. Clark. Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic. 1991.

A Application-Layer Benefits

Many applications benefit from accurate information about path throughput and delay. For example, recently there has been a surge of interest in video streaming, where one of the primary challenges is in estimating the correct bitrate to use. A low estimate hurts video quality while a high estimate risks experiencing a stall in playback. Most algorithms tend to under-estimate rates because stalls hurt the quality of experience more. That, in turn, means they are unable to effectively obtain the true usable path rate.

We showed how every measurement of the queuing delay provides a new estimate of the target rate. Hence, to understand what throughput and delay can be expected from a path, an endpoint only needs to transmit a few packets. The expected performance can be calculated from the measured RTT and queuing delay. These packets can be small, containing only the header and no data, which reduces the bandwidth consumed by probes. Applications can use this information in many ways.

Copa offers a way for applications to obtain rate information. Senders can use the techniques we have developed to measure "expected throughput" – i.e., the rate that a Copa sender will use – by sending only a few small packets, and take an informed decision regarding what quality of content to transfer. As shown in §5.1, Copa's rate estimates are accurate and senders are able to jump directly to the correct rate.

Quick estimation of a transport protocol's expected transmission rate is also useful for selecting good paths or endpoints. For instance, peer-to-peer networks can regularly send tiny packets without payload to monitor the throughput and delay available on the link to a peer. The monitoring is inexpensive, but can enable more informed decisions. Content Distribution Networks using Copa for data delivery can use this too, by routing requests to the appropriate servers. For instance, they can route to minimize the flow-completion time estimated as $2{\cdot}\mathrm{RTT}+l/\lambda$, where l is the flow length and λ is the rate estimate.