Copa: Practical Delay-Based Congestion Control for the Internet

Presenter: Haoran Wan

Drawback of Previous Schemes

- Packet loss (or ECN) based algorithms (Reno, Cubic, etc.): network buffers are filled – **high queuing delay**. (May not be true with AQMs)
- Delay based algorithms (Vegas, FAST): prone to overestimate delay due to ACK compression, network jiter – **underutilize the link**.
- Special cases of network (cellular, web, WiFi): lack of **generalizability**.
- Objective-optimization methods (Remy, PCC, Vivace): online rules are much more **complex** and hard for humans to reason about.

Copa's High Level Goal

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 is also **simple to understand** and is **general in its applicability** to a wide range of environments and workloads;
- And performs at least **as well as the best prior schemes** designed for particular situations.

Problem Definition

Optimization goal:

$$U = \log \underbrace{\boxed{\lambda}}_{\text{Average throughput}} - \underbrace{\boxed{\delta}}_{\text{Weight}} \log \underbrace{\boxed{d}}_{\text{Packet delay}}$$

Under certain simplified (but reasonable) modeling assumptions of packet arrivals, the steady-state sending rate that maximizes U is:

$$\lambda = \frac{1}{\delta \cdot \boxed{d_q}} \quad \text{Mean per-packet queuing delay}$$

Why? Explained later.

Copa Algorithm

Incorporate three ideas:

1. A target rate to aim for;
2. A window update rule that depends moves the sender towards the target rate;
3. A TCP-competitive strategy to compete well with buffer-filling flows.

Target Rate and Update Rule

Congestion window (cwnd), sender estimates the current rate:

$$\lambda = cwnd/\boxed{RTTstanding}$$

Smallest RTT observed over a recent time-window.

The time-window t=srtt/2, where srtt is the current value of the standard smoothed RTT estimate. Why not latest RTT sample? Avoid RTT compression and jitters.

RTTstanding is the RTT corresponding to a “standing” queue, thus the queuing delay is calculated as:

$$d_q = RTTstanding - \boxed{RTTmin}$$

Smallest RTT observed over a long time-window.

Long time window= min(10s, session period).

Target Rate and Update Rule

With the dq calculated, we can calculate the target rate according to the previous equation:

$$\lambda = \frac{1}{\delta \cdot d_q}$$

If current rate exceeds the target, the sender reduce cwnd; otherwise, it increases cwnd.

Sender paces packets at a rate of 2*cwnd/RTTstanding, to avoid packet burst.

Target Rate and Update Rule

To wrap up, upon receiving ACK, Copa applies the following steps:

1. Update the queuing delay d_q using RTTstanding and RTTmin, using the standard TCP EWMA estimator;
2. Set $\lambda_t = 1/(\delta \cdot d_q)$
3. If $\lambda = cwnd/RTTstanding \leq \lambda_t$, then

   $$cwnd = cwnd + v/(\delta \cdot cwnd)$$

   , where v is a "velocity parameter". Otherwise,

   $$cwnd = cwnd - v/(\delta \cdot cwnd)$$

4. The velocity parameter, v, speeds-up convergence....

Target Rate and Update Rule

4. Once per window, the sender compares the current cwnd to the cwnd value at the time that the latest acknowledged packet was sent. If the current cwnd is larger, then set direction to “up”; if it is small, then set direction to “down”. Now if the direction is the same as in the previous window, then double v.

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.

⬆⬆ / ⬇⬇ : v = 2* v   ⬆⬇ : v = 1

Competing with Buffer-Filling Schemes

Modify Copa to compete well with buffer-filling algorithms such as Cubic and NewReno while maintaining its good properties. 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.

Competing with Buffer-Filling Schemes

A detector to detect whether Copa is competing with buffer-filling schemes.

Key Copa property:

  the queue is empty at least once every 5 RTT. (explained later)

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*.

“Nearly empty”: any queuing delay lower than 10% of the rate oscillations in the last 4 RTTs:

$$d_q < 0.1(\boxed{RTTmax} - RTTmin)$$

Max RTT over last 4 RTTs.

Competing with Buffer-Filling Schemes

In competitive mode the sender varies $1/\delta$ according to whatever buffer-filling algorithm one wishes to emulate (e.g., NewReno, Cubic, etc.).

In competitive mode, $\delta \leq 0.5$. When Copa switches from competitive mode to default mode, it resets δ to 0.5.

Even competing with buffer-filling flow, the queue length could be low (when a loss happens to other flows), Copa will switch between default mode and competitive mode.

Dynamics of Copa

Assertion: In steady state, each Copa flow makes small oscillations about the target rate, which also is the equilibrium 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** 0 and $2.5/\hat{\delta}$ packets every 5 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.

Dynamics of Copa

Assumption: link rate is constant u, feedback delay is constant RTTmin ≅ RTT, meaning the queue length inferred from an ACK at time t is :

$$q(t) = \boxed{w(t - RTTmin)} - \boxed{BDP}$$

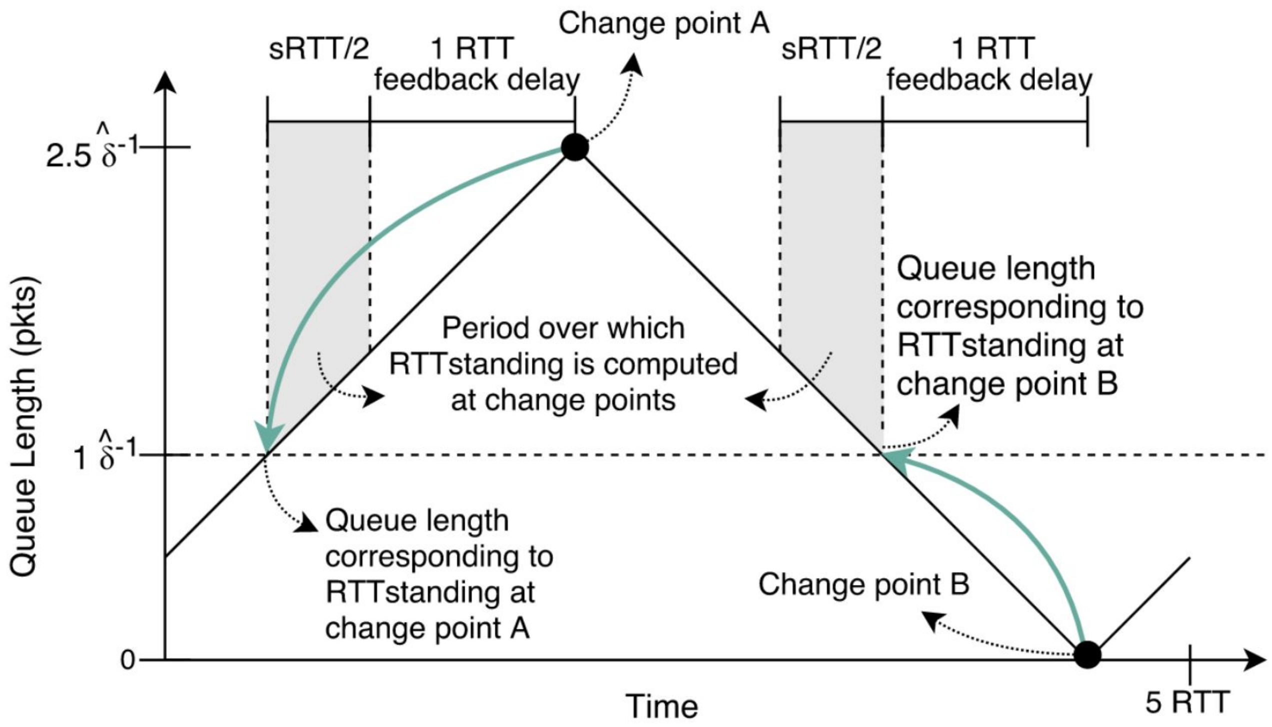
w(t) is congestion window at time t

Bandwidth delay product, in-flights bits.

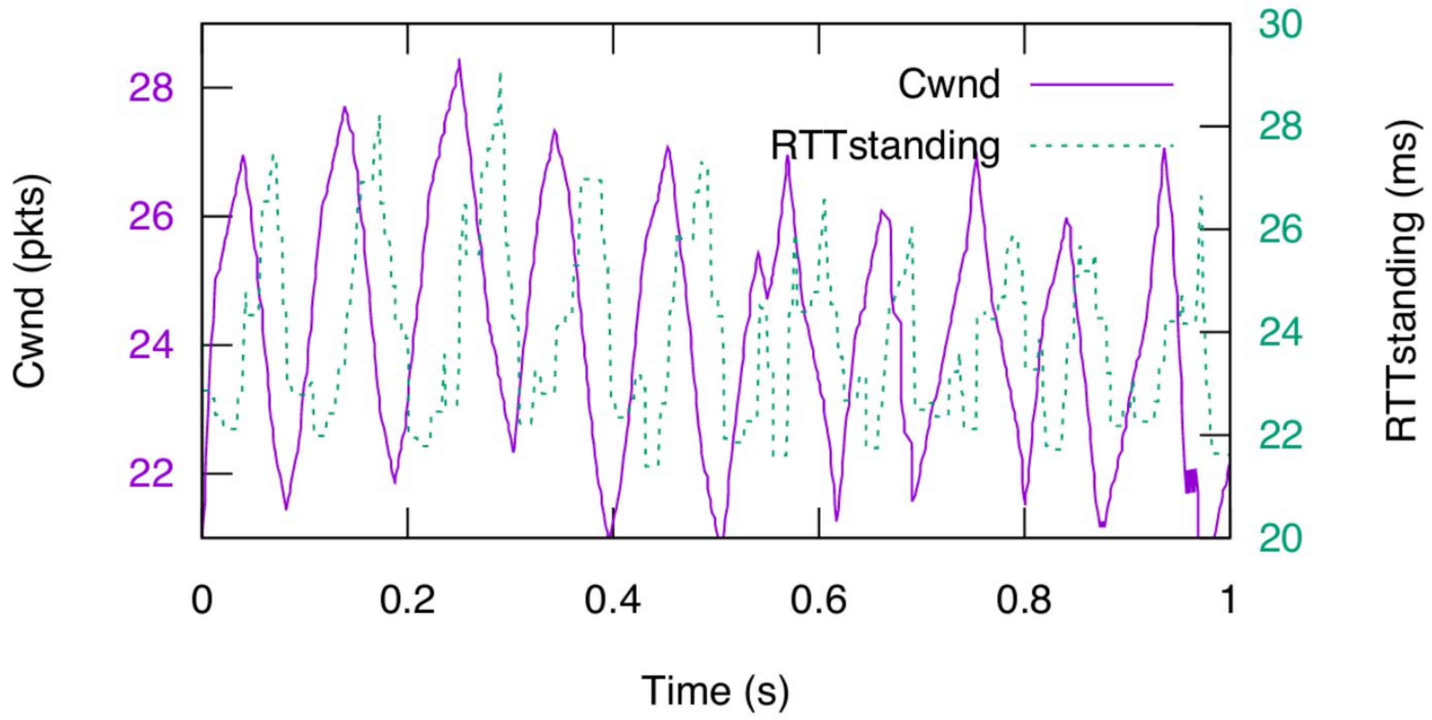
Under the constant-delay assumption, the sending rate is:

$$cwnd/RTT = cwnd/RTTmin$$

Dynamics of Copa



Dynamics of Copa



Justification of the Target Rate

After all the analysis above, we still don't know how they decide the target rate:

$$\lambda = \frac{1}{\delta \cdot d_q}$$

Consider the objective function for sender i:

$$U_i = \log \lambda_i - \delta_i \log \boxed{d_s}$$

Switch delay: $d_s = d_q + 1/\mu$

Suppose each sender attempts to maximize its own objective function, the system will reach a Nash equilibrium when no sender can increase its objective function by unilaterally changing its rate.

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

Objective Function and Nash Equilibrium

Lemma 1. Consider a network with n flows, with flow i sending packets with rate λi 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,

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

where

$$\hat{\delta} = \left(\sum 1/\delta_i\right)^{-1}$$

Objective Function and Nash Equilibrium

Proof. Denote the total arrival rate as $\lambda = \sum_j \lambda_j$, for an M/M/1 queue, the sum of the average wait time in the queue and the link is $\frac{1}{\mu-\lambda}$, given the dequeue rate is μ. Then we got ($d_s = \frac{1}{\mu-\lambda}$):

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

Setting the partial derivative $\frac{\partial U_i}{\partial \lambda_i} = 0$, we have:

$$\frac{\partial U_i}{\partial \lambda_i} = \frac{1}{\lambda_i} - \frac{\delta_i}{\mu - \sum_j \lambda_j} = 0$$

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

Second derivative, $-1/\lambda_i^2 - \delta_i/(\mu - \lambda)^2 < 0$, so it's a maximum point.

Objective Function and Nash Equilibrium

So each flow i should have the sending rate, which:

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

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

The unique solution of this family of linear equations is:

$$\lambda = \sum \lambda_i = \frac{\mu}{(1+\hat{\delta})}$$

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

which is the desired equilibrium rate of sender i

Update Rule Follows from the Equilibrium Rate

At equilibrium rate, the inter-send time between packets is:

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

As previously discussed, the switch delay for the M/M/1 queue is :

$$d_s = \frac{1}{\mu-\lambda}$$

Then the inter-send time became:

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

Properties of the Equilibrium

$$\lambda = \sum \lambda_i = \frac{\mu}{(1+\hat{\delta})}$$

Meaning the equilibrium queuing delay is $1 + 1/\hat{\delta}$, if all $\lambda_i = 0.5$, the number of packets enqueued is 2n+1.

If all node has the same δ, it's equivalent to dividing the capacity between n senders and δ "pseudo-senders", since $\lambda_i = \mu/(\delta + n)$.

If we want low latency, we can increase δ.

Evaluation

User-space implementation for Copa, and it’s compared with linux kernel Cubic, Vegas, Reno and BBR, user-space implementations of Remy, PCC, Vivace, Sprout, and Verus.

Ns-2 simulation: compared with Cubic, NewReno, Vegas, DCTCP.

Evaluation

A 100 Mbit/s link with 20 ms RTT and 1 BDP buffer using Linux qdiscs.

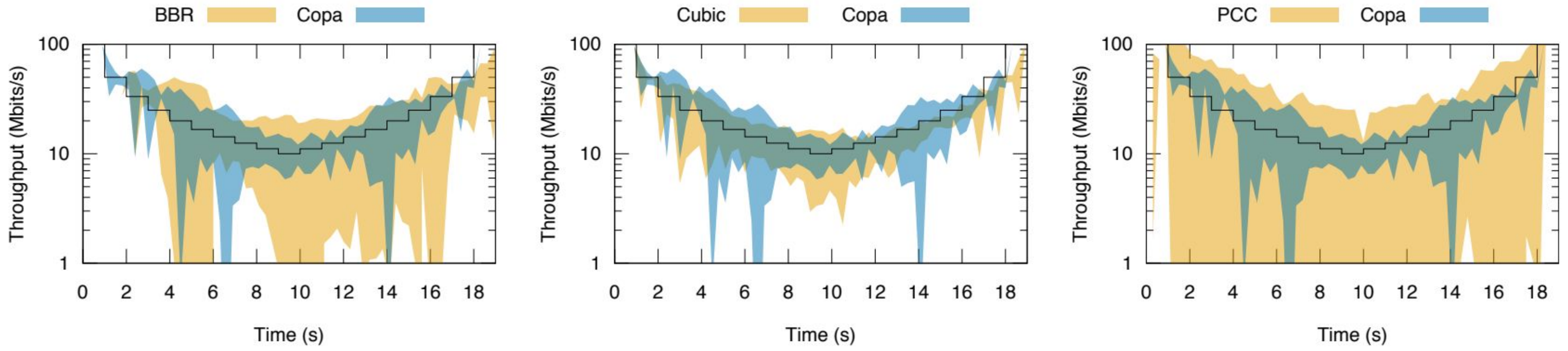


Figure 3: Mean ± std. deviation of throughput of 10 flows as they enter and leave the network once a second. The black line indicates the ideal allocation. Graphs for BBR, Cubic and PCC are shown alongside Copa in each figure for comparison. Copa and Cubic flows follow the ideal allocation closely.

Evaluation

$$J = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \times \sum_{i=1}^{n} x_i^2}$$

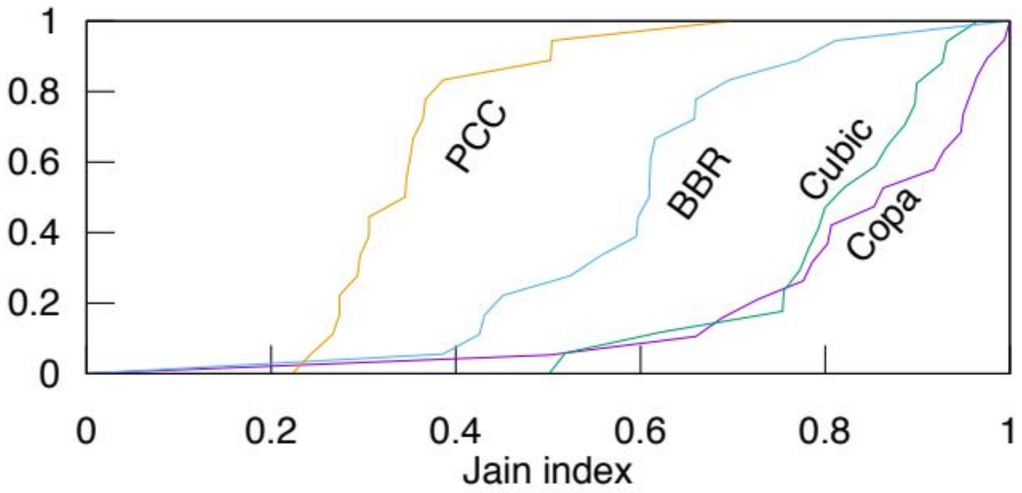
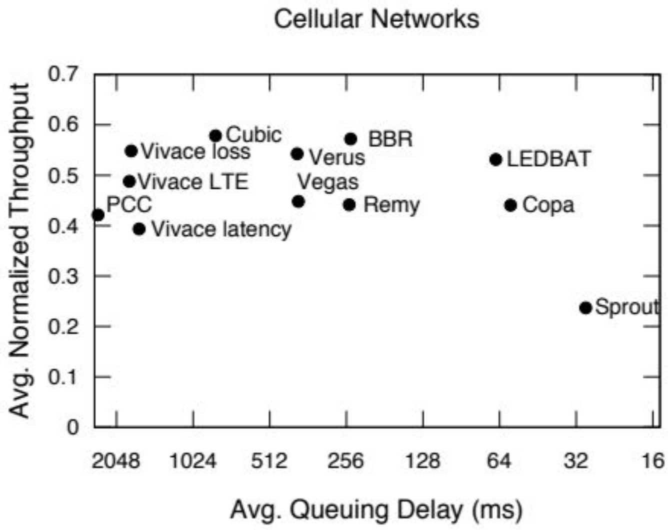
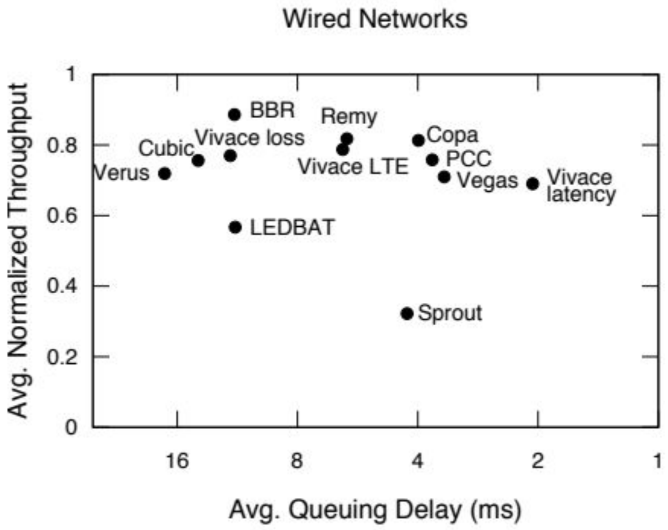


Figure 4: A CDF of the Jain indices obtained at various timeslots for the dynamic behavior experiment (§5.1)

Evaluation

Real-world pantheon evaluation. 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.





Evaluation

Fairness on RTT

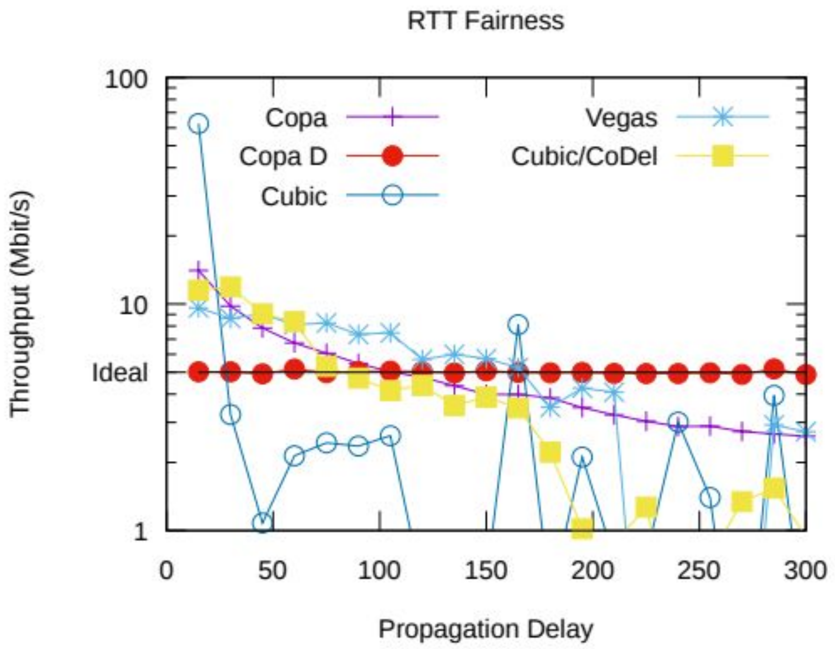


Figure 6: RTT-fairness of various schemes. Throughput of 20 long-running flows sharing a 100 Mbit/s bottleneck link versus their respective propagation delays. "Copa D" is Copa in the default mode without mode switching. "Copa" is the full algorithm. "Ideal" shows the fair allocation, which "Copa D"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

Evaluation

Robustness to packet loss.

Datacenter environment.

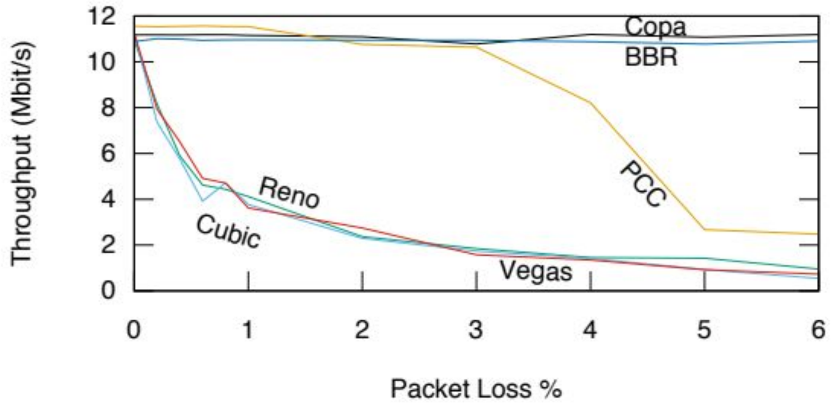


Figure 7: Performance of various schemes in the presence of stochastic packet loss over a 12 Mbit/s link with a 50 ms RTT.

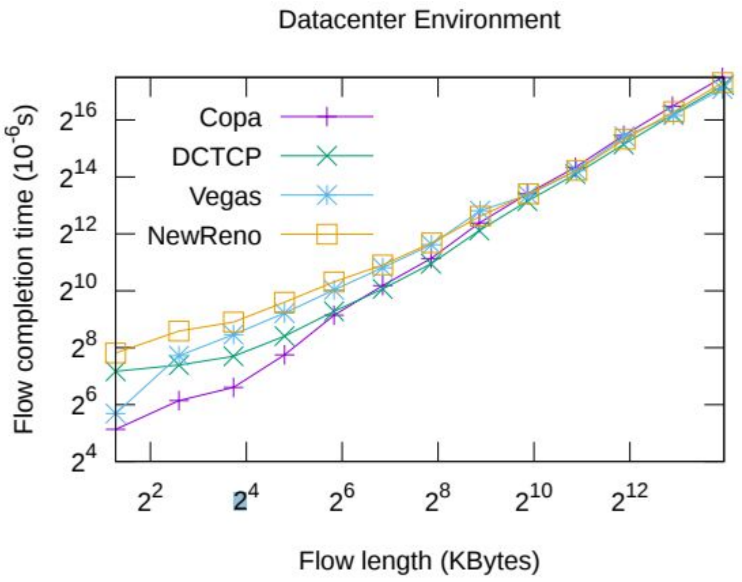


Figure 8: Flow completion times achieved by various schemes in a datacenter environment. Note the log scale.

Evaluation

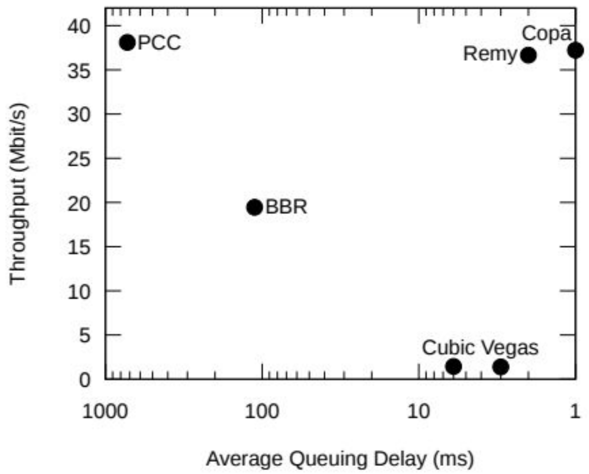


Figure 9: Throughput vs. delay plot for a satellite link. Notice that algorithms that are not very loss sensitive (including PCC, which ignores small loss rates) all do well on throughput, but the delay-sensitive ones get substantially lower delay as well. Note the log-scale.

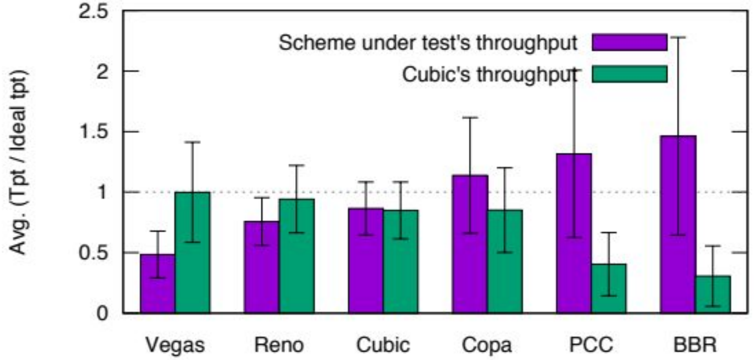


Figure 10: Throughput of different schemes versus Cubic shown by plotting the mean and standard deviation of the ratio of each flow's throughput to the ideal fair rate. The mean is over several runs of randomly sampled networks. The left and right bars show the value for the scheme being tested and Cubic 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.

Opinion

Copa’s scientific part is well formulated and the evaluation result looks great.

Copa doesn’t act directly on the bottleneck queue, but use the delay to optimize the queue delay.

Wonder how it will perform under some Active Queue Management schemes.

Discussion

https://app.perusall.com/courses/cos597s_f2024-advanced-topics-in-computer-science-recent-advances-in-wireless-networks/copa?assignmentId=a4GiTy4QckQKd68M3&part=1