CS 244 Project Report: Reproducing Copa

Sawyer Birnbaum, Nadin El-Yabroudi
Stanford University
sawyerb@stanford.edu,nadin@stanford.edu

ABSTRACT

We reproduce key findings from *Copa: Practical Delay-Based Congestion Control*, which presents a general purpose delay-based congestion control algorithm called Copa. In keeping with the original paper, we find that Copa achieves a higher flow-rate fairness than BBR, PCC, and Cubic. We also successfully reproduce the paper's claim that on real-world wired and cellular links in February of 2018 Copa achieves nearly as much throughput and 2-10 times lower queuing delay as other congestion control schemes. However, our results show that in the last 2 months Copa achieves significantly lower throughput than most other schemes, and on in our fairness experiments we observe that a single Copa flow uses only about 85% of a link's full capacity. Accordingly, our results indicate that further research is required to establish whether and in what conditions Copa provides better performance than other congestion control algorithms.

1 INTRODUCTION

The authors of *Copa: Practical Delay-Based Congestion Control for the Internet*[2] suggest a new congestion control algorithm, Copa, that they claim can simultaneously achieve high throughput, low queuing delay, and flow rate fairness. The algorithm optimizes an objection function that combines a flow's average throughput and packet delay:

$$U = \log \lambda - \delta \log d$$

where λ is the throughput, d is the end-to-end packet delay, and δ is a parameter that determines how much to weigh delay compared with throughput. The authors set $\delta = 0.5$. The authors show that under a Markovian packet arrival model, the value of λ that maximizes U is:

$$\lambda = \frac{1}{\delta \dot{d}_q}$$

where d_q is the mean per-packet queuing delay. This value of λ is the target that Copa attempts to achieve by updating its sending window and estimating the queuing delay. Specifically, on each ACK arrival, Copa adjusts the congestion window, cwnd, as follows:

(1) Update an estimate of the queuing delay d_q.
(2) Set $\lambda = 1/(\delta \cdot d_q)$.
(3) If cwd/RTTstanding $\leq \lambda$, then increase the cwnd by $v/(\delta * cwnd)$. Otherwise decrease cwnd by the same amount. Here RTTstanding is the smallest RTT observed over a recent time interval τ and v is the "velocity parameter".
(4) Update v based on the number of consecutive times that cwnd has changed in the same direction. This speeds up convergence.

More detail on Copa's update process can be seen in Figure 1. Note that the bottleneck queue length oscillates around δ^{-1} packets. As the queue length increases the so does d_q, prompting a reduction of cwnd. This in turn causes the queue to empty, reducing d_q and triggering an increase of cwnd.

The authors provide no explanation for their choice of objective function and spend little time justifying their decision to set $\delta = 0.5$ and their assumption of Markovian packet arrival. They argue that that "[u]ltimately, our validation of the Copa algorithm is through experiments." Accordingly, we attempted to verify two important experimental results from [2], namely the following:

- Copa outperforms other congestion control algorithms on real-world networks on cellular and wired links by achieving nearly as much throughput and 2-10 times lower queuing delay; and
- Copa maintains nearly full link utilization with a median Jain fairness index of 0.93. The median indices for Cubic, BBR and PCC are 0.90, 0.73 and 0.60 respectively.

We chose to reproduce these particular experimental results because they are central to the authors' claims that Copa achieves high throughput, low queuing delay, and flow rate fairness.

Evaluating these claims entails reproducing Figures 4 and 5 from the original paper. To reproduce the real-world network traffic experiment, we analyzed data from the Stanford Congestion Control Pantheon [4] during the time when the authors did their research and in the past two months. We found that Copa did outperform other congestion control algorithm on cellular and wired networks at the time when the authors conducted their research but that in the last few months Copa achieves much lower throughput than most other algorithms on wired links. To reproduce the Jain Fairness Index we used an Mininet emulated link with the same characteristics as those used by the authors. We used the authors' implementation of Copa to run the same fairness experiment. Our results show that Copa achieves a median Jain index of 0.97, Cubic a median of 0.88, BBR a median of 0.61, and PCC a median of 0.81. [1]

2 PRIOR WORK

Three decades of congestion control research have given rise to many types of congestion control schemes. The original end-to-end congestion control algorithm in TCP uses packet loss as its congestion control signal, as did many schemes that followed it. Because these algorithms fill up network buffers, they achieve high throughput at the expense of high queuing delay. To address this problem, a separate thread of research approximates delay and uses this as a signal for congestion control. This branch of research includes such protocols as Vegas[6] and FAST[3]. However, network jitter and ACK compression can cause overestimates of the delay, resulting in under-utilized links. As a result, newer congestion control

[1] The code we used to run our reproduction is available at https://github.com/nadinelyab/Copa_Reproduction.

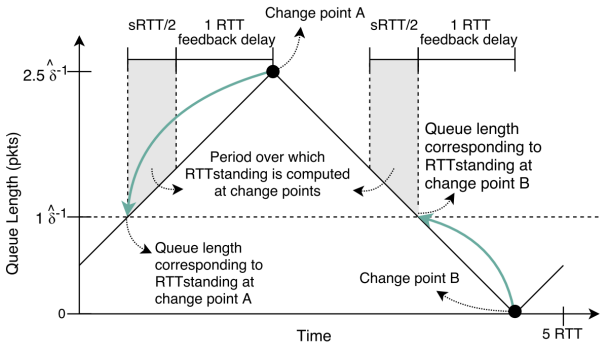

Figure 1: One Copa cycle: Evolution of queue length with time. This is Figure 1 of [2]

schemes are often specialized to a particular type of network or use-case, such as cellular networks[5, 13], datacenters [7, 8], or video streaming[11, 14]. Other new congestion control algorithms (e.g., Remy [1, 12], PCC[10], and PCC Vivace[9]) argue that the space of congestion control signals and actions is too complicated for human engineering, and instead apply online learning to develop policies for responding to the current observed network behavior. These newer schemes either trade interpretability for performance or specialize in achieving the metrics necessary for their intended application area.

Copa [2] belongs to the line of algorithms that use an estimate of delay to update the congestion window and modulate the sending rate. One of the weaknesses of these schemes is that they do not interact well with loss-based protocols when both exist on a network. The delay-based algorithms tend to be less aggressive. Copa attempts to solve this problem by detecting the presence of buffer-fillers and changing the δ parameter in the presence of such flows so that it also applies an additive-increase/multiplicative-decrease approach. Furthermore, unlike newer schemes, Copa aims to be *practical* meaning that it can be used in many environments including cellular and wired links and it is relatively simple to understand compared with online learning schemes.

3 EXPERIMENTS

3.1 Pantheon

The authors of Copa submitted their implementation of Copa to Pantheon [4] to compare Copa with other congestion control schemes on real-world Ethernet and cellular links. They plotted the averaged normalized throughput and average queuing delay for each congestion control algorithm. The average normalized throughput is measured by normalizing relative to the flow with the highest throughput for one experiment and then averaging across all runs and flows in that experiment. The average queuing delay is calculated by subtracting the minimum delay for a flow from all other delay measurements in that flow. The experiments chosen from Pantheon [4] each last 30 seconds. Half of them only had one flow, and the other half had three flows each starting at 0, 10, and 20 seconds from the start of the experiment. Figure 2 shows the results in [2].

For our reproduction, we analyzed results from Pantheon [4] at the time when the authors ran their experiments as well as in March and April of 2019. We reached out to the authors and learned they used data from before February 21st, 2018 and after February 19th, 2018. Therefore, we used data from Pantheon from February 20th, 2018 which represented five different countries. The authors claim to use data from six different countries but we found that no experiments for one country were available in the dates provided to us. This left us with 15 experiments for wired links. Additionally, there were no results from cellular links between February 20th and 21st on Pantheon, so we chose 32 cellular experiments from before February 19th. We then chose the 40 wired link experiments between March 26th and April 24th of 2019 and the 34 cellular links between January 21st and April 23rd of 2019 that were available on Pantheon to see if Copa still performed better than other congestion control algorithms. We wrote our own scripts which used the packet-by-packet log files available on Pantheon to calculate the average queuing delay as described in [2] and we used the summarized results given in Pantheon to calculate the normalized throughput for each experiment.

3.2 Jain's Fairness

Jain's fairness index is a measure of the fairness of a resource allocation scheme. The index is defined as follows, where x_i denotes the resources allocated to party i:

$$\mathcal{J}(x_1, x_2, ..., x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \sum_{i=1}^{n} x_i^2} \tag{1}$$

In this context, x_i represents the throughput of flow i. When all flows receive the same throughput the index is maximized with a value of 1. When one flow receives all of the throughput, the index is minimized with a value of $\frac{1}{n}$.

To compare the fairness of congestion control protocols, the authors run the following experiment:

(1) Every second for 10 seconds, start flow f_i.
(2) After 10 seconds, every second for 10 seconds end flow f_{10-i}.

Thus, flow 1 runs for 20 seconds, flow 2 for 18 seconds, and so on. The authors measure the throughput of each flow, compute the Jain's fairness index for every millisecond of the experiment, and plot a CDF of the indices for four congestion control protocols: Copa, Cubic, BBR, and PCC. They run the experiment over a 46 Mbit/s Mahimahi link with 20ms RTT and 1 BDP of buffer.

Figure 3 presents the results of the experiment. Copa is the fairest protocol, followed by Cubic, BRR, and finally PCC. Copa achieves a median Jain fairness index of 0.93 while Cubic, BBR and PCC achieve median indices of 0.90, 0.73 and 0.60, respectively.

We reached out to the authors, and they shared with us their scripts for running this experiment and plotting the data[3]; we used these scripts as a basis for our reproduction. To create Copa flows, the script uses the authors' GenericCC[4] implementation of their algorithm, and to create Cubic and BBR flows it uses the Linux kernel implementations of those algorithms. We modified the provided code in three ways. First, instead of Mahimahi, we use a Mininet

[3]https://github.com/venkatarun95/CopaEvaluation; see `experiment-dynamic.sh` and `pcap-tpt-graph.py`
[4]https://github.com/venkatarun95/genericCC

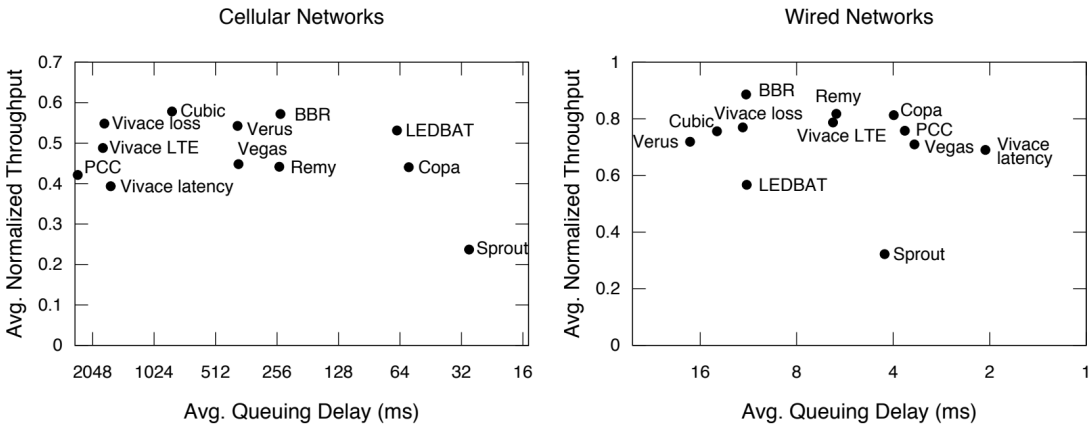

Figure 2: Original Real-World Experiments on Pantheon paths for 6 different countries. This was Figure 5 in [2]

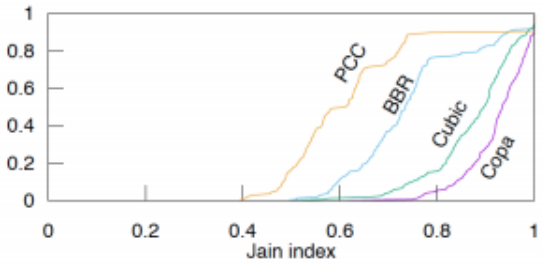

Figure 3: Original CDF of the Jain Indices obtained in the Copa Paper. This was Figure 4 in [2]. [2]

emulator (with the same bandwidth, RTT, and buffer settings as in the original experiment). Second, because the authors' code for establishing PCC flows referenced some files that they did not provide, we used the PCC developers' implementation to run these flow. (The Copa authors also used the PCC developers' implementation.) And third, to resolve a bug we experienced while running the authors' plotting script, we modified two lines of that file. (We explained these changes to the authors; they did not indicate that these modifications would interfere with our visualization of the results.)

4 RESULTS

4.1 Pantheon

After running our original experiments we found that for wired links many congestion control algorithms were not achieving the same results as [2]. Therefore, we reached out to the authors again and learned that they had ignored flows for which the throughput was greater than 120 Mb/s since the Copa algorithm implementation on Pantheon does not scale well to high throughput. We corrected our scripts to ignore these flows and obtained results much closer to the original paper. Figure 4 and 5, show the comparison of the original results with the results of our reproduction for cellular and wired links. Although there are differences, likely due to the fact we do not know which exact experiments were used in producing the original figure, the general claims of the paper hold. We find that:

- For wired links, Copa achieves nearly as much throughput as other schemes like BBR and about 3 times lower queuing delay.
- For cellular links, Copa's throughput is a lower than most other congestion control algorithms but not by much. Copa's queuing delay is in some cases 30 times lower than other congestion control algorithms. However LEDBAT performs better than Copa, achieving more throughput and only a fraction of higher queuing delay.

We were also curious to see if other congestion control algorithms not present in the original Pantheon plot could outperform Copa. Therefore, we ran the same experiments on all congestion control algorithms and present the results in Table 1. We find that for wired links Indigo achieves the highest throughput with a delay only slightly higher than Copa, yet Indigo did not appear in [2]. For cellular links LEDBAT achieves the best results with comparable throughput to other schemes and 2-4 times better queuing delay.

For March and April of 2019 we found that the claims of the paper do not hold. The results for these experiments are shown in Figures **??** and 10. For wired links, Copa achieves significantly lower throughput than all congestion control algorithms except Sprout. However, Copa does achieve queuing delay 2-10 times lower than other congestion control algorithms. For cellular links, Copa's throughput is more comparable to other congestion control schemes but it is still lower than most others. Copa's delay is still 2-10 times lower than other congestion control algorithms. This is likely due to several factors such as changes in network behavior over the past year or changes in Pantheon such as the implementation and availability of other congestion control algorithms as well as the addition of a new node in Saudi Arabia.

Scheme	Wired Links		Cellular Links	
	Throughput	Delay	Throughput	Delay
Copa	0.702	4.27	0.410	70.75
Fillp	0.798	16.10	0.680	2721
QUIC	0.648	11.17	0.660	818.7
Sprout	0.312	3.455	0.220	40.30
Vivace Latency	0.608	2.441	0.440	1808
Vivace Loss	0.655	11.25	0.615	1981
Vivace LTE	0.669	7.01	0.563	1845
SCREAM	0.003176	0.450	0.066	20.47
Vegas	0.660	3.44	0.494	334.9
BBR	0.741	13.47	0.664	360.6
Verus	0.601	12.92	0.626	473.6
PCC	0.626	6.194	0.501	2401
Indigo	0.758	5.120	0.346	48.31
WebRTC	0.0226	1.377	0.195	120.7
Taova	0.701	8.159	0.537	285.3
LEDBAT	0.538	6.194	0.579	83.24

Table 1: Reproduction results for all congestion control algorithms in Pantheon February 2018. Delay is in milliseconds and throughput in Mbit/s.

4.2 Jain Fairness

Figure 6 presents the results of our fairness experiment. In keeping with the authors' findings, we observe that Copa is the fairest algorithm, followed by Cubic. However, in contrast with the original results, we find that PCC is a significantly fairer protocol than BBR. The authors speculate that this stems from improvements to the PCC algorithm made since the the original experiment was run. Additionally, excepting PCC, the Jain indices we observe vary slightly for Copa and Cubic and significantly for BBR. In our experiment, Copa achieves a median Jain index of 0.97, Cubic a median of 0.88, BBR an median of 0.61, and PCC a median of 0.81. For Copa and Cubic, this constitutes an increase of 0.03 and a decrease of 0.02 over the median values in the original paper, respectively. For BBR, this constitutes 0.12 decrease over the original median value. We reached out to the authors about these differences; they noted that our results are qualitatively similar to the original ones. It possible that the difference in BBR fairness stems from changes to the Linux Kernel implementation of BBR, although it would be somewhat surprising if those changes made the algorithm less fair.

Figure 11 (see appendix) displays the throughput of each flow. As expected, the Copa flows clearly exhibit the most fair behavior.

We noticed that for some of the congestion control protocols, namely Cubic and BBR, flows do not start and end in exactly 1 second intervals, perhaps because of some delays in the `iperf` program used to establish these flows or in the kernel implementations of these protocols. Accordingly, we ran another experiment in which we increased the inter-flow time (i.e., the time between consecutive flow starts and flow ends) from 1 second to 5 seconds. By spacing the flows farther apart, we reasoned that the variance in exact flow start and end times would matter less. Figure 7 presents the results of this experiment. Overall, the results are consistent with those form the earlier experiment, with the exceptions that the PCC protocol now slightly edges out the Cubic protocol and that the BBR algorithm preforms much better. The median indices are 0.98 for Copa, 0.90 for Cubic, 0.81 for BBR, and 0.94 for PCC. (See Figure 7 in the appendix for throughput plots for this experiment; unlike in the original experiment, Cubic and BBR flows enter and exit at approximately the correct interval.)

To verify that the paper's results do not depend on the specific conditions of the emulated link, we also experimented with modifying the link's bandwidth, RTT, and queue size. Additionally, we collected data using the Reno and Vegas congestion control algorithms. Although the performance of some congestion control protocols varied in these tests, Copa consistently achieved the fairest behavior. See Figure 13 in the appendix for the results of a some of these experiments.

4.3 Throughput

In the process of running the Jain fairness experiment, we noticed that even when a single Copa flow is running, it does not use 100% of the link bandwidth. Although the link supports 46 Mbits/second, during the 1st second of the experiment, the lone Copa flow runs at only about 35 Mbits/second. Moreover, we continue to observe this behavior on the 5 second inter-flow test, which should provide the single flow enough time to reach an equilibrium state. In contrast, individual flows using the other protocols do achieve the max throughput.

To investigate further, we ran a single Copa flow over the 46 Mbit/second flow for 10 seconds. Figure 8 displays the throughput of this flow. As in the fairness experiments, it does not utilize the full throughput of the link. The authors speculate that perhaps some difference between the Mininet and Mahimahi emulators is responsible for the sub-100% link utilization. They also note (without elaboration) that their GenericCC implementation of Copa does not scale well to high throughputs, but that their Linux Kernel implementation of the algorithm does.

5 DISCUSSION

5.1 Implications of Findings

Our findings provide a mixed picture of Copa effectiveness: while we find that the Figures from the original paper can be reproduced with reasonable accuracy and that Copa consistently behaves more fairly than other congestion control algorithms, we also observe that in many cases Copa achieves a low throughput compared with these alternatives. In our reproduction of the Pantheon results at approximately the same time that the authors conducted the original experiment, we record small but meaningful differences in the average delay and throughput of almost every protocol. However, it seems reasonable that that these differences stem from the fact that we do not know exactly which links and what dates were use in original experiment. Moreover, the relative performance of the protocols remains essentially uncharged in our reproduction, so the differences we observe do not undermine the authors' conclusions about Copa's performance.

In contrast, when we reran our analysis on data from the past few months, we found that Copa's relative performance has deteriorated, especially with regards to its throughput on wired networks. Thus, our results indicate that authors' findings are specific only

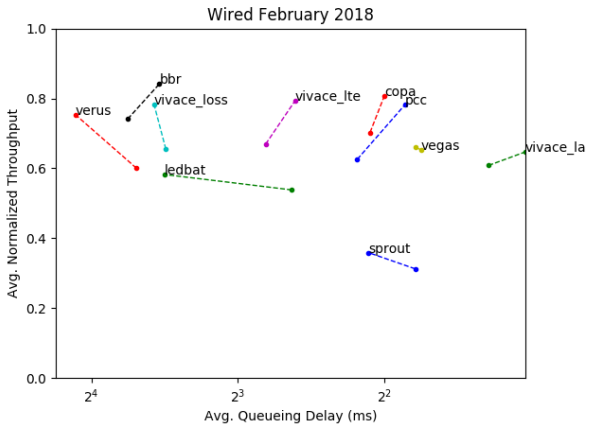

Figure 4: Comparison of original results from [2] and our reproduction for wired links in February of 2018. The label represent the data point in [2]. Each point is connected by a line to the point data point of our reproduction. Some congestion control algorithms like Remy and Cubic are not shown because the information was not available on Pantheon.

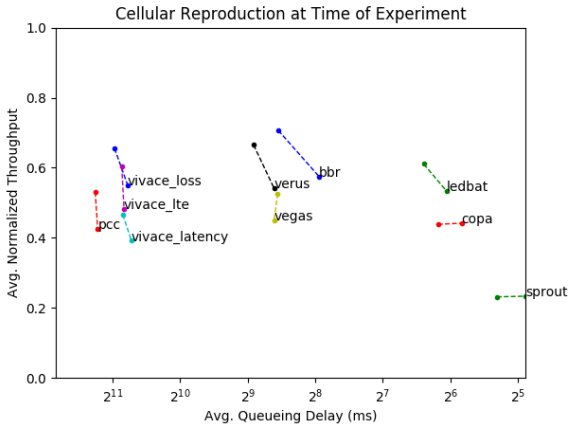

Figure 5: Comparison of original results from [2] and our reproduction for cellular links in February of 2018. The label represent the data point in [2]. Each point is connected by a line to the point data point of our reproduction. Some congestion control algorithms like Remy and Cubic are not shown because the information was not available on Pantheon.

to the network conditions present at the time of their experiment. Future work is required to establish whether the original or new results are more indicative of Copa's potential. (See Future Works section.)

Our reproduction of the Jain fairness experiment support the authors' position that Copa is fairer than many other popular congestion control protocols. Except for PCC, the results in our direct

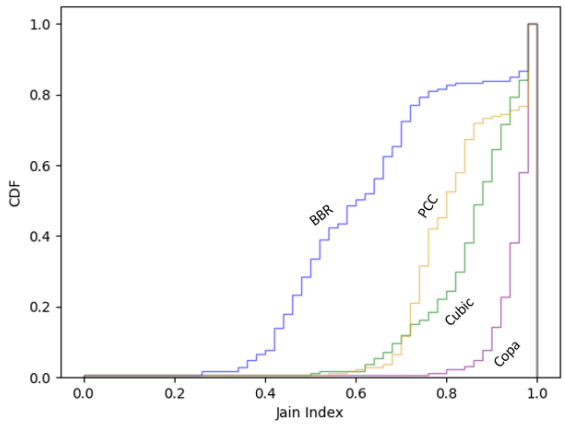

Figure 6: Our reproduction of plot of Jain Indices CDF obtained in the Copa Paper. Note that the CDF is over 1 ms intervals in the experiment.

replication of the experiment in the paper line up reasonably well with the original data. For Copa, Cubic, and (to a lesser extent) BBR, we recorded median Jain index values that differed from the ones in the paper by only a few hundredths, and these difference seem reasonable given our use of a different emulator which might process and send packets at a slightly different rate than the one used by the authors. The larger difference we observed in the performance of the PCC algorithm also makes sense given that we used a more recent version of PCC. Moreover, we found that Copa continues to outperform the other algorithms under a variety of experimental conditions, including with a different inter-flow time and with a different link bandwidth, RTT, and queue size. Copa also consistently outperforms the Vegas and Reno protocols. Overall, these findings provide strong support for the authors' claim that Copa is a very RTT-fair algorithm.

However, in the process of conducting the fairness experiment, we noticed that a single Copa flow does not use 100% of the link bandwidth, and in fact uses only about 85% of the available capacity. (Interestingly, in the authors' description of the fairness experiment, they note that Copa "maintains *nearly* full link utilization." Italics added.) This seriously undermines the authors' finding that Copa achieves comparable throughput to other congestion control algorithms. It also raises questions about the reproducibility of some of the other experiments in the paper which show Copa using the entire link capacity. (See the Future Work section for more details.)

Overall, our findings suggest that Copa provides lower delay and better flow-rate fairness than other congestion control protocols, but that this comes at the cost of reduced throughput. It is possible that changing the δ parameter in favor of throughput maximization could alter this balance and allow Copa to achieve better throughput (see Future Work section). Regardless, however, our findings indicate that optimizing Copa's performance requires careful tuning of this parameter and suggest that Copa is not an ideal congestion control algorithm under some reasonably common network conditions.

5.2 Limitations and Weaknesses

Limited Information about Pantheon Experiment Data. The Copa paper does not specify the dates or Pantheon links used to collect throughput and delay information for Figure 5. We contacted the authors for more information, but they were unable to provide us with list of the exact dates and links they used. They did inform us the they collected data between February 20 and 21, 2018 and that they did not use any links with a max throughput above 120 Mbit/s. We were also not able to obtain the raw data points for the original Pantheon graphs. Therefore, we attempted to estimate their values from the figure in order to compare these results with our own.

Mahimahi Emulator. The original fairness experiment ran on a Mahimahi emulator. We attempted to use a Mahimahi emulator but experienced difficulties configuring the emulator with the bandwidth, RTT, and queue settings used by the authors. Accordingly, we switched to a Mininet emulator. In theory, the results should not be affected by our choice of emulator.

6 FUTURE WORK

With more time, we could continue to explore the conditions under which Copa achieves higher throughput, lower delay, and fairer behavior compared with other congestion control protocols and could study the extent to which Copa achieves the optimal sending rate (according to its packet arrival modeling assumptions). Specifically, we could run the following experiments:

(1) Investigate Copa's throughput to better understand why the algorithm sometimes uses less than 100% of the available bandwidth. This could involve reproducing Figure 7 from the original paper, which indicates that Copa does achieve max throughput when running on a 12 Mbit/s link.
(2) Explore Copa's fairness in a more complex environment with flows running different congestion control algorithms. Specifically, we could reproduce Figure 10 from the original paper, which presents the results of an experiment in which the authors run Copa flows alongside Cubic flows.
(3) Measure the affect of altering the δ parameter. The authors set δ to 0.5; it would be interesting to observe how throughput and delay across an emulated link change as this parameter varies.
(4) Investigate how closely Copa matches the target sending rate λ. To do so, we could measure the the per-packet queuing delay d_q for a Copa flow running on an emulated link and measure the λ chosen by Copa based on its estimation of d_q. This would test the effectiveness of Copa's d_q estimation algorithm and window update strategy.
(5) Analyze Pantheon data across a longer time interval and track the performance of Copa on a weekly or monthly basis. We could also compare Copa's performance across Pantheon links and determine the link qualities that correlate with better performance of the algorithm.

A more ambitious goal would be to make small changes to the utility function (e.g., let $U = \lambda + \delta d$), solve for the optimal λ given a Markovian packet arrival, and measure the extent to which these changes affect algorithm's performance.

7 CONCLUSION

We reproduced several key experiments in *Copa: Practical Delay-Based Congestion Control*. We found that at the time when Copa was created in February of 2018, it outperformed other congestion control algorithms on both wired and cellular real-world networks. We also found that Copa consistently achieves a higher Jain Fairness index than PCC, BBR, and Cubic across a range of link conditions. However, in data from the past 2 months, we observed that Copa achieves significantly lower throughput that most other schemes on wired links, and we observe that a lone Copa flow does not achieve full utilization of a link's capacity. Thus, our results suggest that Copa does fulfill the author's goal of designing a congestion control protocol capable of simultaneously providing high throughput, low delay, and flow-rate fairness. Future work should study the conditions under which Copa's throughput deteriorates and investigate whether the utility function chosen by the authors is the correct objective to maximize.

REFERENCES

[1] P. Thaker A. Sivaraman, K. Winstein and H. Balakrishnan. [n. d.]. An Experimental Study of the Learnability of Congestion Control. ([n. d.]).
[2] Venkat Arun and Hari Balakrishnan. 2018. Copa: Practical Delay-Based Congestion Control for the Internet. *15th USENIX Symposium on Networked Systems Design and Implementation* (2018).
[3] S. Low D. Wei, C. Jin and S. Hegde. [n. d.]. FAST TCP: Motivation, Architecture, Algorithms, Performance. ([n. d.]).
[4] R. S. Wahby Levis F. Y. Yan J. Ma G. Hill, D. Raghavan and K. Winstein. [n. d.]. Pantheon: the training ground for internet congestion-control research. ([n. d.]).
[5] A. Sivararaman K. Winstein and K. Balakrishnan. [n. d.]. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. ([n. d.]).
[6] S. W. O'Malley L. S. Brakmo and L. L. Peterson. [n. d.]. TCP Vegas: New Techniques for Congestion Detection and Avoidance. ([n. d.]).
[7] D. A. Maltz J. Padhye P. Patel B. Prabhakar S. Sengupta M. Alizadeh, A. Greenberg and M. Sridharan. [n. d.]. Data Center TCP (DCTCP). ([n. d.]).
[8] T. Edsall B. Prabhakar A. Vahdat M. Alizadeh, A. Kabbani and M. Yasuda. [n. d.]. Less is More: Trading a Little Bandwidth for Ultra-low Latency in the Data Center. ([n. d.]).
[9] D. Zarchy E. Arslan Y. Gilad B. Godfrey M. Dong, T. Meng and M. Schapira. [n. d.]. Pcc vivace: Online-learning congestion control. ([n. d.]).
[10] D. Zarchy P. B. Godfrey M. Dong, Q. Li and M. Schapira. [n. d.]. PCC: Re-architecting Congestion Control for Consistent High Performance. ([n. d.]).
[11] A. Jain M. Ghobadi, Y. Cheng and M. Mathis. [n. d.]. Trickle: Rate limiting youtube video streaming. ([n. d.]).
[12] K. Winstein and H. Balakrishnan. [n. d.]. TCP ex Machina: Computer-Generated Congestion Control. ([n. d.]).
[13] J. Chen L. Subramanian Y. Zaki, T. Potsch and C. Gorg. [n. d.]. Adaptive congestion control for unpredictable cellular networks. ([n. d.]).
[14] J. Gahm R. Pan H. Hu A. Began Z. Li, X. Zhu and D. Oran. [n. d.]. Probe and adapt: Rate adaptation for HTTP video streaming at scale. ([n. d.]).

8 APPENDIX: ADDITONAL FIGURES

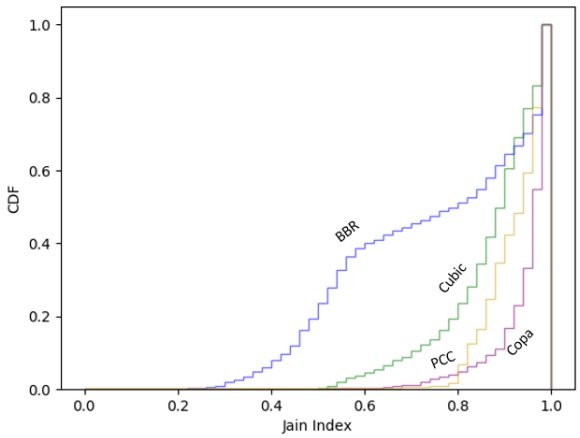

Figure 7: CDF of the Jain Indices obtained with a 5 second inter-flow time.

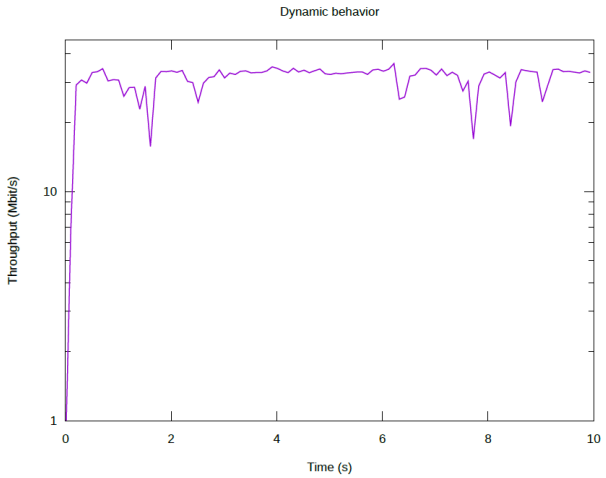

Figure 8: Throughput of a single Copa flow running on a 46 Mbit/second link. Note that the flow does not use the full capacity of the link.

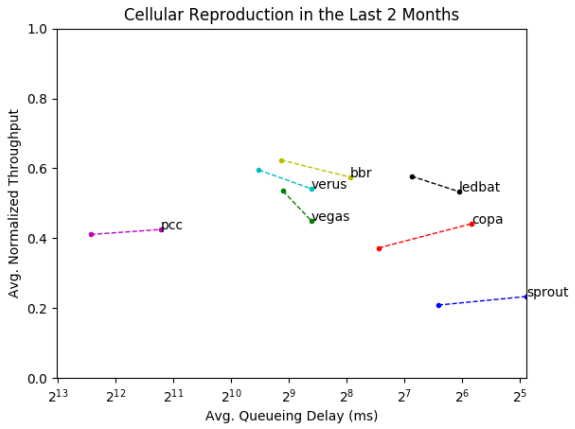

Figure 9: Comparison of original results from [2] and our reproduction for cellular links from January to April of 2019. The label represent the data point in [2]. Each point is connected by a line to the point data point of our reproduction. Some congestion control algorithms like Vivace LTE and Vivace Loss are not shown because they do not exist in Pantheon anymore.

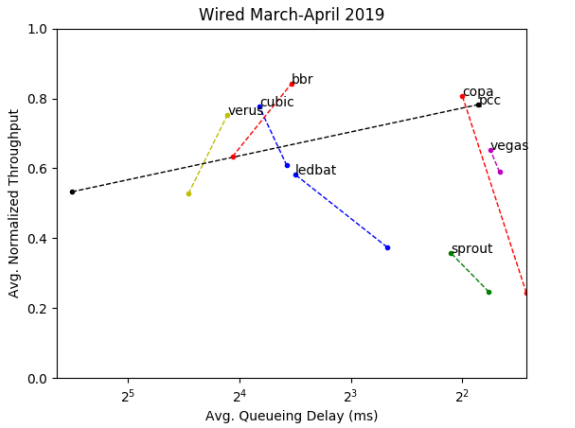

Figure 10: Comparison of original results from [2] and our reproduction for wired links in March and April of 2019. The label represent the data point in [2]. Each point is connected by a line to the point data point of our reproduction. Some congestion control algorithms like Vivace LTE and Vivace Loss are not shown because they do not exist in Pantheon anymore.

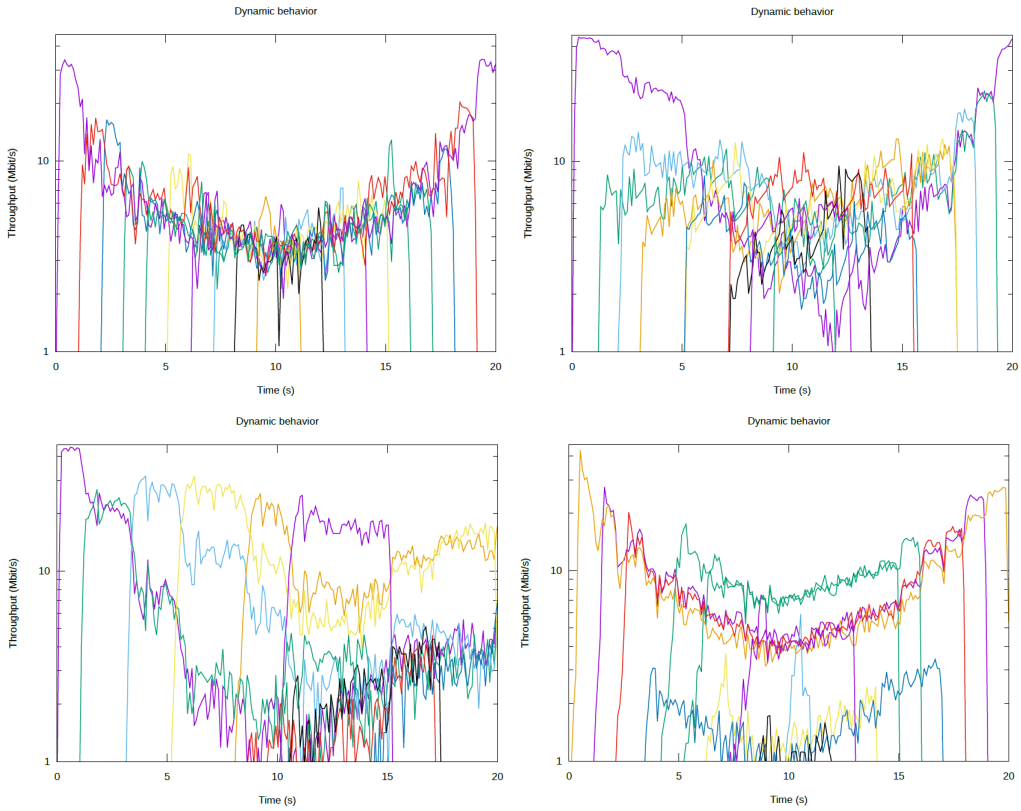

Figure 11: Flow throughputs in our reproduction of the Jain fairness experiment. The Copa flows behave the fairest. *Top Left*: Copa | *Top Right*: Cubic | *Bottom Left*: BBR | *Bottom Right*: PCC

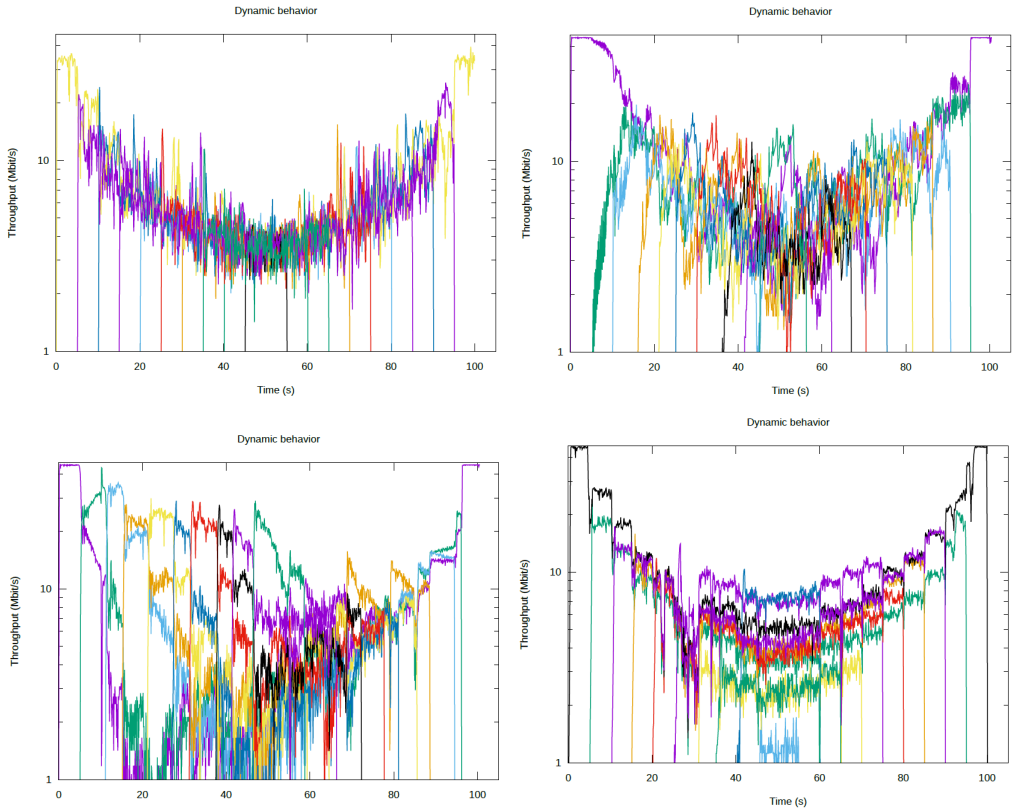

Figure 12: Flow throughputs in our reproduction of the Jain fairness experiment with a 5 second inter-flow time. Note that now the Cubic and BBR flows begin and end at approximately the correct times. *Top Left*: Copa | *Top Right*: Cubic | *Bottom Left*: BBR | *Bottom Right*: PCC

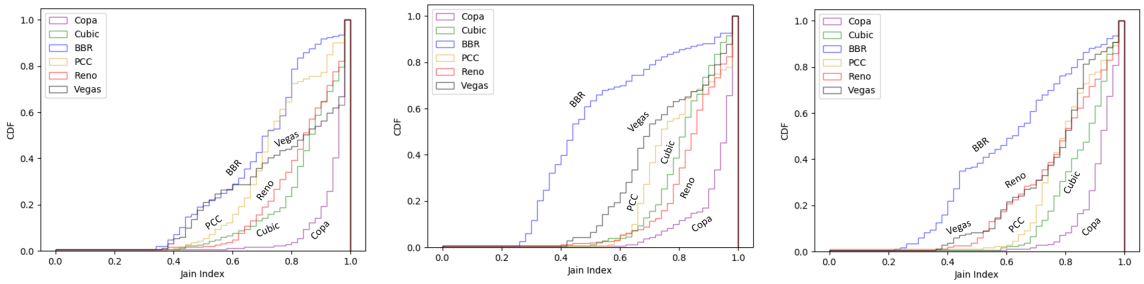

Figure 13: CDFs of Jain indices from experiments with different link parameters. Copa is consistently the fairest algorithm. *Left*: 96 Mbit/s bandwidth | *Center*: 40 ms RTT | *Right*: 2 BDP queue size