# Competing TCP Congestion Control Algorithms over a Satellite Network

Pinhan Zhao
*Worcester Polytechnic Institute*
Worcester, MA, USA
azphme@gmail.com

Benjamin Peters
*Worcester Polytechnic Institute*
Worcester, MA, USA
btpeters@wpi.edu

Jae Chung
*Viasat*
Marlborough, MA, USA
jaewon.chung@viasat.com

Mark Claypool
*Worcester Polytechnic Institute*
Worcester, MA, USA
claypool@cs.wpi.edu

***Abstract*—Understanding how new TCP congestion control algorithms interact with the default TCP Cubic over a wide-range of network conditions is important for moving congestion control research forward. Unfortunately, lacking are studies over actual satellite Internet networks where high latencies pose challenges to TCP performance. This paper presents results from experiments over a commercial satellite Internet link assessing TCP congestion control algorithm performance for Cubic when competing with algorithms using four different approaches: loss-based (Cubic), bandwidth-estimation based (BBR), utility function-based (PCC) and satellite optimized (Hybla). Analysis shows: 1) the default Cubic algorithms are fair to each other; 2) Cubic dominates PCC during steady state; 3) Hybla dominates Cubic during start-up; and 4) BBR dominates Cubic during both start-up and steady state.**

## I. Introduction

TCP congestion control algorithms are critical for many network links and most Internet connections in order to achieve high utilization while avoiding network overload. While TCP has been the dominant Internet transport protocol for some time [1], the congestion control algorithms deployed by TCP have evolved. Currently, TCP Cubic [2] is the default congestion control algorithm for most major operating systems (e.g., in Linux as of 2006, Apple MacOS as of 2014, and Microsoft Windows as of 2016), but there are dozens of alternatives proposed to improve performance. However, many congestion control algorithms are designed and developed without considering their performance alongside other congestion control algorithms – of particular importance is how new algorithms contend with TCP Cubic over the wide-range of network conditions in use.

Our specific interests are for satellite Internet networks, an essential part of modern network infrastructures given their ubiquitous network connectivity for remote areas and especially in emergencies when traditional (i.e., wired) connections may not be available. The number of satellites in orbit is over 2100, a 67% increase from 2014 to 2019 [3], and recent research has improved satellite transmission capacities more than 20x, to a total planned capacity for geostationary satellites of over 5 Tb/s. While throughput gains for satellite Internet are promising, satellite latencies remain a challenge. The physics involved for round-trip time Internet communication between terrestrial hosts using a geostationary satellite accounts for about 550 milliseconds of latency at a minimum [4].

Despite their importance, there are few published studies measuring network performance over actual satellite Internet networks [5], with most studies just using either simulation [6] or emulation with satellite parameters [7], [8]. While our previous work has assessed TCP congestion control in isolation [9], what is needed to advance congestion control algorithm research are studies of the *interaction* between TCP congestion control algorithms over satellite Internet, particularly alternate algorithms competing for bandwidth with TCP Cubic.

This paper presents results from experiments that measure the performance of TCP congestion control algorithms – default loss-based (Cubic [2]), bandwidth estimation-based (BBR [10]), utility function-based (PCC [8]), and satellite-optimized (Hybla [11]) – over a commercial satellite Internet network. We consider the aforementioned four algorithms (Cubic, BBR, PCC and Hybla) competing with TCP Cubic, a common network default. The network testbed and experiments are done over the Internet but designed to be as comparable across runs as possible by interlacing runs of each protocol serially to minimize temporal differences and by doing 10 bulk downloads for each test condition to provide for a large sample.

Analysis of the results shows Cubic is fair when competing with another Cubic flow over our satellite Internet link, equally sharing the network in all phases of a download. In contrast, BBR dominates Cubic owing to BBR's disregard for packet losses that restrict Cubic's throughput. In turn, Cubic dominates PCC, likely because the latter reduces data rates with latency. Hybla dominates Cubic in the start-up phase since it more quickly ramps up data rates with high latency, but is fair during steady state. Overall, the BBR-Cubic throughput differences are the largest (about a 100 Mb/s difference out of 120 Mb/s total), with the Hybla-Cubic start-up differences a close second (about 80 Mb/s out of 120 Mb/s).

The rest of this report is organized as follows: Section II describes research related to this work, Section III describes our testbed and experimental methodology, Section IV analyzes our experiment data, and Section V summarizes our conclusions and suggests possible future work.

## II. Related Work

This section describes work related to our paper, including TCP congestion control (Section II-A), comparisons of TCP

congestion control algorithms (Section II-B), and TCP performance over satellite networks (Section II-C).

### A. TCP Congestion Control

There have been numerous proposals for improvements to TCP's congestion control algorithm since its inception. This section highlights a few of the papers most relevant to our work, presented in chronological order.

Caini and Firrincielli [11] propose TCP Hybla to overcome the limitations traditional TCP flows have when running over high-latency links (e.g., a satellite). TCP Hybla modifies the standard congestion window increase with: a) an extension to the "additive increase"; b) adoption of the SACK option and timestamps [12], which help in the presence of loss, and c) using packet spacing to reduce transmission burstiness. The slow start (SS) and congestion avoidance (CA) algorithms for Hybla are:

$$SS : cwnd = cwnd + 2^{\rho} - 1 \tag{1}$$

$$CA : cwnd = cwnd + \frac{\rho^2}{cwnd} \tag{2}$$

$$\rho = \frac{RTT}{RTT_0} \tag{3}$$

where $RTT_0$ is typically fixed at a "wired" round-trip time of 0.025 seconds. Hybla is available for Linux as of kernel 2.6.11 (in 2005).

Ha et al. [2] develop TCP Cubic as an incremental improvement to earlier congestion control algorithms. Cubic is less aggressive than previous TCP congestion control algorithms in most steady-state cases, but can probe for more bandwidth quickly when needed. Cubic's window size is dependent only on the last congestion event, providing for more fairness to flows that share a bottleneck but have different round-trip times. TCP Cubic has been the default in Linux as of kernel 2.6.19 (in 2007), Windows 10.1709 Fall Creator's Update (in 2017), and Windows Server 2016 1709 update (in 2017).

Cardwell et al. [10] develop TCP Bottleneck Bandwidth and Round-trip time (BBR) as an alternative to Cubic. BBR uses the maximum bandwidth and minimum round-trip time observed over a recent time window to build a model of the network and set the congestion window size (up to twice the bandwidth-delay product). BBR has been deployed by Google servers since at least 2017 and is available for Linux TCP since Linux kernel 4.9 (end of 2016).

Dong et al. [8] propose TCP PCC that continuously observes performance based on measurements in the form of mini "experiments." Different actions taken in these experiments are compared using a utility function, adopting the rate that has the best performance. The authors compare PCC against other TCP congestion control algorithms, including Cubic and Hybla, and emulate a satellite network based on parameters from a satellite Internet system. PCC is not generally available for Linux, but we were able to obtain a Linux-based implementation from Compira Labs.[1]

[1]https://www.compiralabs.com/

### B. Comparison of Congestion Control Algorithms

Cao et al. [13] analyze measurement results of BBR and Cubic over a range of different network conditions. They produce heat maps and a decision tree that identifies conditions which show performance benefits from BBR over using Cubic. They find it is the relative difference between the bottleneck buffer size and bandwidth-delay product that dictates when BBR performs well.

Ware et al. [14] model how BBR interacts with loss-based congestion control protocols (e.g., TCP Cubic). Their validated model shows BBR becomes window-limited by its in-flight cap which then determines BBR's bandwidth consumption. Their models allow for predictions of BBR's throughput when competing with Cubic with less than a 10% error.

Turkovic et al. [15] study the interactions between congestion control algorithms. They measure performance in a network testbed using a "representative" algorithm from three main groups of TCP congestion control – loss-based (TCP Cubic), delay-based (TCP Vegas [16]) and hybrid (TCP BBR) – using 2 flows with combinations of protocols competing with each other. They also do some evaluation of QUIC [17] as an alternative transport protocol to TCP. They observe bandwidth fairness issues, except for Vegas and BBR, and find BBR is sensitive to even small changes in round-trip time.

### C. TCP over Satellite Networks

Obata et al. [5] evaluate TCP performance over actual (not emulated, as is typical) satellite networks. Specifically, they compare a satellite-oriented TCP congestion control algorithm (STAR) with TCP NewReno and TCP Hybla. Experiments with the Wideband InterNetworking Engineering test and Demonstration Satellite (WINDS) system show throughputs around 26 Mb/s and round-trip times around 860 milliseconds. Both TCP STAR and TCP Hybla have better throughputs over the satellite links than TCP NewReno.

Wang et al. [18] provide a preliminary performance evaluation of QUIC with BBR on a network testbed that emulates a satellite network (capacities 1 Mb/s and 10 Mb/s, RTTs 200, 400 and 1000 ms, and packet loss rates up to 20%). Their results confirm QUIC with BBR has some throughput improvements compared with TCP Cubic for their emulated satellite network.

Utsumi et al. [7] develop an analytic model for TCP Hybla for steady-state throughput and latency over satellite links. They verify the accuracy of their model with simulated and emulated satellite links (capacity 8 Mb/s, RTT 550 ms, and packet loss rates up to 2%). Their analysis shows substantial improvements to throughput over that of TCP Reno for loss rates above 0.0001%

Our work extends the above with the study of interactions between 4 congestion control algorithms – Cubic, BBR, Hybla and PCC – on an actual, commercial satellite Internet network.

## III. Methodology

In order to evaluate competition between TCP congestion control algorithms over a satellite Internet link, we use the

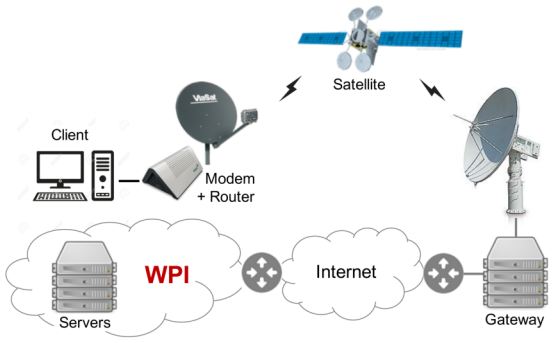



Fig. 1. Satellite measurement testbed.

following methodology: setup a testbed (Section III-A), bulk-download data using each congestion control algorithm versus default TCP Cubic (Section III-B), and then analyze the results (Section IV).

### A. Testbed

We set up a satellite link and configure our clients and servers so as to allow for repeated tests. Our setup is designed to enable comparative performance measurements by keeping all conditions the same across runs as much as possible, except for the change in TCP congestion control algorithm.

Our testbed is depicted in Figure 1. The client is a Linux PC with an Intel i7-1065G7 CPU @ 1.30GHz and 32 GB RAM. There are two servers, the first configured to use the default TCP Cubic and the second configured to use one of: TCP Cubic, TCP BBR, TCP Hybla or TCP PCC. Each server has an Intel Ken E312xx CPU @ 2.5 GHz and 32 GB RAM. The servers and client all run Ubuntu 18.04.4 LTS, Linux kernel version 4.15.0. The servers connect to the WPI LAN via Gb/s Ethernet. Our university (WPI) campus network is connected to the Internet via several 10 Gb/s links, all throttled to 1 Gb/s.

The client connects to a Viasat satellite terminal (with a modem and router) via a Gb/s Ethernet connection. The terminal communicates through a Ka-band outdoor antenna (RF amplifier, up/down converter, reflector and feed) through the Viasat 2 satellite[2] to the larger Ka-band gateway antenna. The terminal supports adaptive coding and modulation using 16-APK, 8 PSK, and QPSK (forward) at 10 to 52 MSym/s and 8PSK, QPSK and BPSK (return) at 0.625 to 20 MSym/s.

The connected Viasat service plan provides a peak data rate of 144 Mb/s. Given minimum satellite round-trip times and the peak data rate, the bandwidth-delay product (BDP) of our satellite link is approximately 140 Mb/sec $\times$ 0.6 sec $=$ 10.5 MBytes.

The gateway does per-client queue management for traffic destined for the client, where the queue can grow up to 36 MBytes allowing a maximum queuing delay of about 2 seconds at the peak data rate. Queue lengths are controlled by Active Queue Management (AQM) that starts to randomly drop incoming packets when the queue grows over half of the limit (i.e., 18 MBytes).

Wireshark captures all packet header data on each server and the client.

The performance enhancing proxy (PEP) that Viasat deploys by default is disabled for all experiments in order to assess the performance of the TCP congestion control algorithms and not the specific PEP implementation.

### B. Downloads

We compare the performance of the four congestion control algorithms (Cubic, BBR (version 1), PCC and Hybla) competing with the default TCP Cubic by using bulk-downloads via `iperf`[3] (v3.3.1) for each flow. Cubic, BBR and Hybla are used without further configuration. PCC is configured to use the Vivace-Latency utility function [19].

For all hosts, the default TCP buffer settings are changed on both the server and client so that flows are not flow-controlled and instead are governed by TCP's congestion window. These include setting `tcp_mem`, `tcp_wmem` and `tcp_rmem` to 60 MBytes, all well above the BDP.

The client initiates connections to both servers via iperf, starting simultaneous downloads for 120 seconds. After the downloads complete, the client continues to the next test condition. After cycling through each condition, the client pauses for 1 minute. The cycle runs a total of 10 times – thus, providing 10 network traces of 2 minutes for each test condition pair over the satellite link. We analyze results from a weekday in May 2021.

Our previous work [20] (focused on performance of non-competing flows) assessed baseline performance for the link (i.e., without any added traffic) and shows the vast majority (99%) of round-trip times are from 560 and 625 milliseconds (median 597 ms, mean 597.5 ms, standard deviation 16.9 ms), and average packet loss rates are about 0.05%, with most of these (77%) single-packet losses.

## IV. Analysis

This section first presents comparative TCP download performance for a single TCP Cubic flow competing with another flow consider throughput, delay (round-trip time), and loss (retransmissions) [21]. We summarize with fairness and throughput differences.

### A. Cubic versus Cubic

We start with TCP Cubic competing with TCP Cubic as a baseline for comparative performance. Cubic is the default congestion control algorithm in Linux and Windows (see Section II-A), and infers that segments needing retransmission are lost and a signal of congestion.

Figure 2 depicts the results. The stacked graphs are aligned temporally on the x-axis based on the time since the session started. From the top-down, the graphs are throughput, round-trip time, congestion window (`cwnd`) and retransmission percent. Each line is the mean across 10 runs, computed once per

[2]https://en.wikipedia.org/wiki/ViaSat-2

[3]https://software.es.net/iperf/

second, with the shading showing 95% confidence intervals around the means.

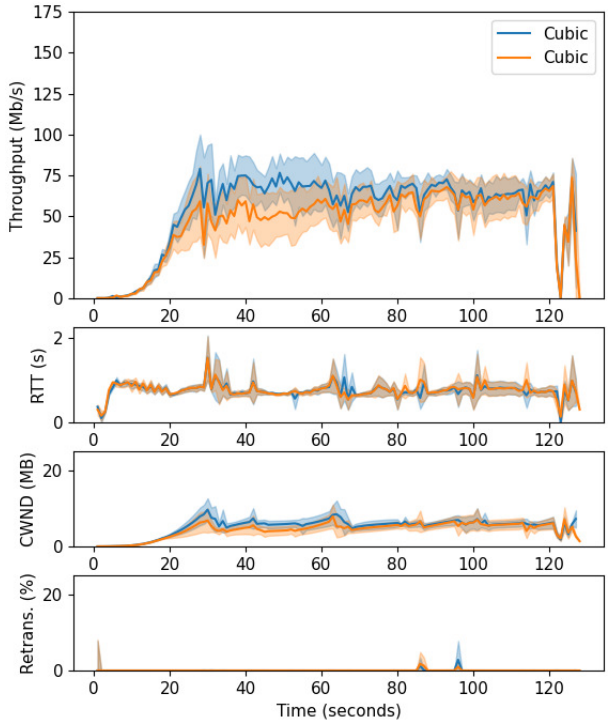



Fig. 2. Cubic versus Cubic

From the graphs, both Cubic throughputs closely follow the same pattern through the first 20 seconds whereupon the second TCP flow (orange) gets somewhat less throughput on average until about 60 seconds in when both flows again get about the same throughput. At all times, the confidence intervals overlap significantly. Both flows experience the same round-trip times since they have the same bottleneck, with times around 0.7 seconds but with peaks around 1 second in some cases. After start-up, the congestion windows are each about 1/2 the BDP – about 5 MBytes. Retransmission rates are negligible.

### B. Cubic versus BBR

In contrast to Cubic, BBR – widely deployed by Google servers and an option for QUIC [22] – uses an estimate of the available bandwidth and round-trip time to determine the size of the congestion window, ignoring packet losses. Given the BDP of about 10.5 MBytes, but a queue limit only of 1.5x this, our previous work [23] would estimate TCP BRR would get considerably more throughput than a competing TCP Cubic. Figure 3 depicts this scenario. The graphs are as for Figure 2, but here the orange lines represent BBR.

From the graphs, BBR dominates throughput from 15 seconds on, getting 10x more bandwidth. This is reflected in the CWND growth. Round-trip times are fairly high for both, ranging from around 0.8 to 1.2 seconds, suggesting considerable queuing at the bottleneck. Cubic's low throughput is likely due to its response to intermittent loss, evidenced by the retransmission spikes in the bottom graph.

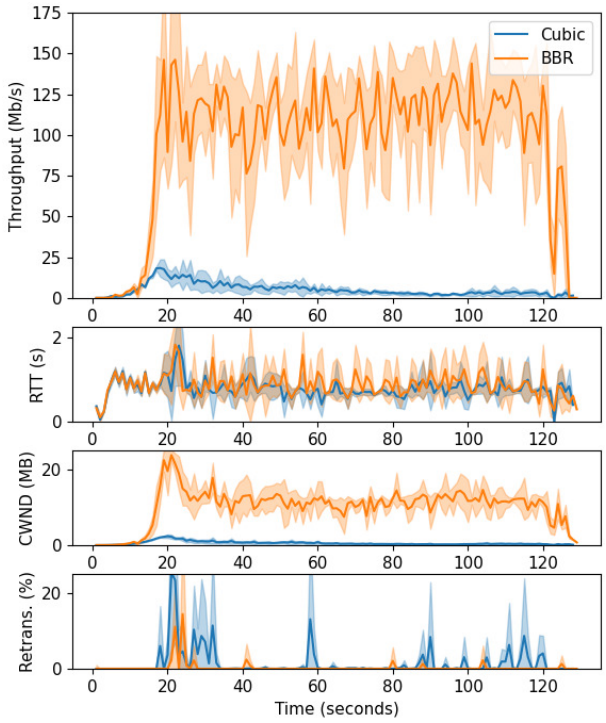



Fig. 3. Cubic versus BBR

Note the widths of the BBR confidence intervals suggest considerable variation in bitrates over time, but this is mostly due to the timing across runs, when BBR probes for minimal round-trip times by drastically reducing bitrates for a few round-trip times [10].

During start-up, Cubic and BBR appear to share the bottleneck equally with average throughputs not diverging until about 15 seconds from the start.

### C. Cubic versus PCC

PCC deploys a utility function to deciding the sending rate based on throughput, loss and round-trip time. Figure 4 shows TCP Cubic versus TCP PCC, with graphs as for Figure 2 and the orange lines representing PCC.

After starting up, Cubic consistently gets about 2x more throughput than PCC, reflected in the congestion window values, too. These differences may be because PCC is prioritizing low delay, so reduces its rate, even though the queue is shared by both flows and Cubic does not consider delays. Round-times are consistently around 0.9 seconds, but rise to as high as 1.3 seconds.

During start-up, Cubic and PCC appear to share the bottleneck equally with average throughputs not diverging until about 15 seconds from the start.

Note the widths of the confidence intervals for PCC, suggesting considerable variation in bitrates from run to run.

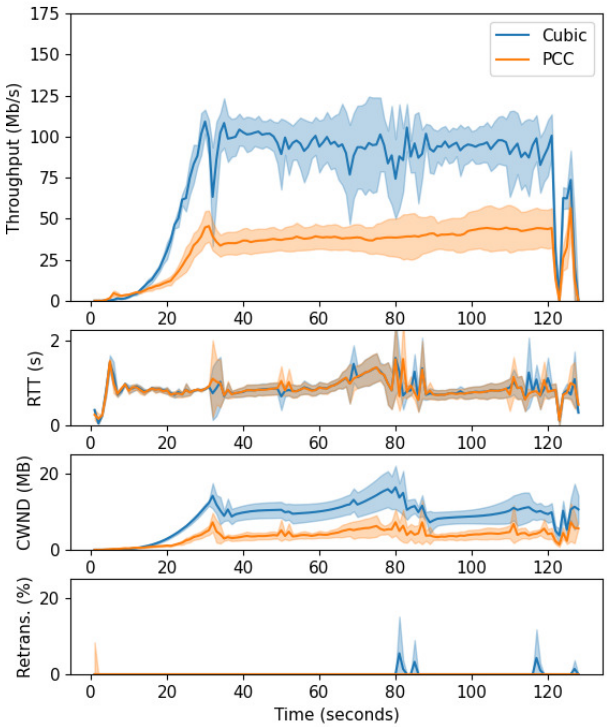



Fig. 4. Cubic versus PCC

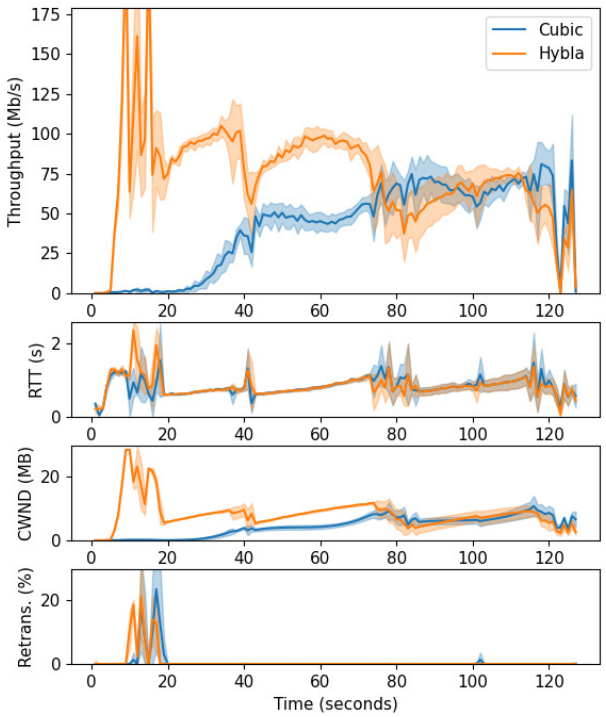



Fig. 5. Cubic versus Hybla

### D. Cubic versus Hybla

Hybla is a latency-optimized version of TCP Reno that is tuned to high-latency conditions (e.g., satellite links), adjusting the congestion window size based on the measured round-trip time compared to a baseline (0.25 seconds by default). Figure 5 shows TCP Cubic versus TCP Hybla, with graphs as for Figure 2 and the orange lines representing Hybla.

From the graphs, Hybla dominates throughput versus Cubic for the first 70 seconds of the download. From time 80 to 120 seconds, however, throughputs are fairly equal. Round-trip times are relatively low compared to the Cubic vs. Cubic baseline, but show periods of steady increase from 0.6 seconds to just over a second. The initial start-up phase for Hybla causes considerable retransmissions and round-trip times over 2 seconds. After that, retransmissions are minimal for both flows.

### E. Cubic without Hystart

Our preliminary evaluation [24] has shown Cubic with Hystart [25] enabled (the default for all TCP connections in Linux since about 2010) can cause TCP over a satellite Internet link to exit slow start prematurely before the congestion window has a chance to reach the approximate maximum. We evaluate how TCP Cubic with Hystart enabled competes with TCP Cubic with Hystart disabled.[4]

Figure 6 shows the same scenario as Figure 2, but the orange flow is TCP Cubic with Hystart off. From the graphs, the flow with Hystart off gets far more throughput during start-up, with this advantage persisting until around 45 seconds in whereupon the flows have comparable throughputs. This difference is likely because Hystart causes Cubic to exit slow start prematurely before the appropriate congestion window is reached – the ideal congestion window should be about 5 MBytes for each flow based on the bandwidth-delay product. The higher data rates for Hystart off also cause high round-trip times until about 15 seconds in, but then fairly steady round-trip times around 0.7 seconds.

[4]Disabled via: echo 0 > /sys/module/tcp_cubic/parameters/hystart

### F. Summary

As a summary for performance, we computing Jain's fairness index [26], one of the most widely used metrics for measuring the fairness of system resources allocation. For two flows with throughputs $x_1$ and $x_2$:

$$f(x_1, x_2) = \frac{(x_1 + x_2)^2}{2(x_1^2 + x_2^2)} \tag{4}$$

Jain's fairness ranges from $\frac{1}{2}$ (most unfair) to 1 (most fair). Since TCP congestion control includes at least two distinct congestion response phases (slow start and congestion avoidance), we also consider three different regions: 1) start-up, the first 15 seconds, often before the flow has reached the available bandwidth; 2) steady state, the last 60 seconds of our flows, where a TCP flow operates only at the available bandwidth; and 3) overall, including throughput for the entire 120 second download.

Figure 7a has the fairness results, shown with a heatmap for start-up, steady state and overall for each scenario. The

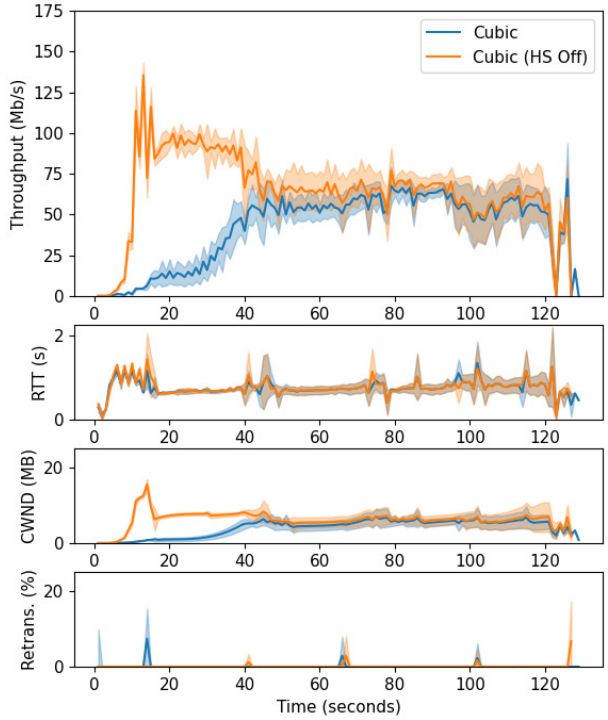
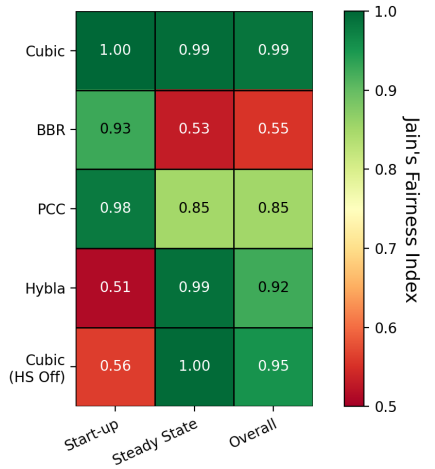
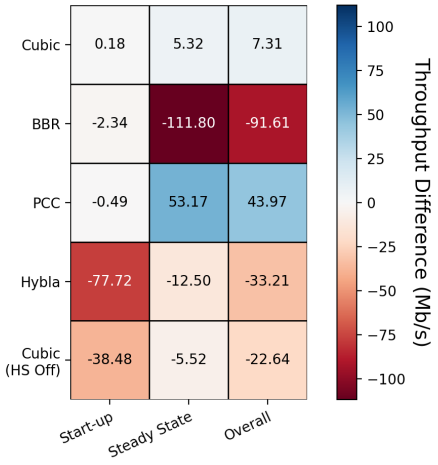



Fig. 6. Cubic (Hystart on) versus Cubic (Hystart off)




(a) Fairness.




(b) Throughput difference.

Fig. 7. Flows competing with Cubic – start-up, steady state and overall.

green shades indicate mid-range or better fairness, whereas red indicates mid-range or worse fairness. Figure 7b shows the throughput differences corresponding to the same scenarios. Here, blue indicates the default Cubic flow gets more throughput and red indicates the competing flow gets more throughput with the color intensity reflecting degree of difference.

From the graphs comparing another flow to Cubic, Cubic is quite fair at all phases. Conversely, BBR is somewhat unfair at start-up and considerably unfair at steady state getting a significantly more bandwidth in the latter case and making it unfair overall. PCC is fair at start-up, but gets considerably lower bandwidth at steady state and overall. Hybla is unfair at start-up, getting a lot more bandwidth at start-up, but quite fair at steady state making it moderately fair overall. Cubic with Hystart off is unfair to Cubic at start-up, getting more bandwidth initially before being fair at steady state.

## V. Conclusion

TCP congestion control algorithms are important for high network utilization while avoiding overload. Understanding interactions between existing congestion control algorithms, particularly TCP Cubic, and new congestion control algorithms is important before widespread deployment. Satellite Internet links pose particular challenges due to their high latencies but are relatively under-studied despite their importance.

This paper presents results from experiments on a production satellite Internet network, considering the competition for the link with default TCP Cubic with 4 congestion control algorithms: loss-based Cubic, bandwidth estimation-based BBR, utility function-based PCC, and satellite optimized Hybla.

Analysis shows Cubic shares the satellite link fairly with itself during both start-up and steady-state. BBR, on the other hand, dominates Cubic in both these same phases. PCC, in contrast, is dominated by Cubic in both phases. Hybla dominates Cubic during start-up but is fair during steady-state and overall. The throughput differences for BBR and Hybla start-up are the largest (about 9x). These insights should be useful for TCP congestion control research moving forward, particularly as algorithms are adjusted over a wide range of link conditions and in conjunction with existing protocols.

There are several areas we are keen to pursue as future work. Other settings to TCP, such as the initial congestion windows, may have a significant impact on performance. Novel utility functions (e.g., for PCC [19]) may have benefits when tuned for high latency. Improvements to BBR startup [27] may benefit BBR over satellite links. The emerging transport protocol QUIC [17] merits evaluation, particularly as it has modular congestion control algorithms (Cubic and BBR) that should be assessed for their interactions with TCP Cubic over satellites.

## ACKNOWLEDGMENTS

Thanks to Amit Cohen, Lev Gloukhenki and Michael Schapira of Compira Labs for providing the implementation of PCC.

## REFERENCES

[1] J. Postel, "Transmission Control Protocol," *IETF Request for Comments (RFC) 793*, Sep. 1981.

[2] S. Ha, I. Rhee, and L. Xu, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, 2008.

[3] Satellite Industries Association, "Introduction to the Satellite Industry," Online presentation: https://tinyurl.com/y5m7z77e, 2020.

[4] Cisco, *Interface and Hardware Component Configuration Guide, Cisco IOS Release 15M&T*. Cisco Systems, Inc., 2015, chapter: Rate Based Satellite Control Protocol.

[5] H. Obata, K. Tamehiro, and K. Ishida, "Experimental Evaluation of TCP-STAR for Satellite Internet over WINDS," in *Proceedings of the International Symposium on Autonomous Decentralized Systems*, Tokyo, Japan, Mar. 2011.

[6] C. Barakat, N. Chaher, W. Dabbous, and E. Altman, "Improving TCP/IP over Geostationary Satellite Links," in *Proceedings of GLOBECOM*, Rio de Janeireo, Brazil,, Dec. 1999.

[7] S. Utsumi, S. Muhammad, S. Zabir, Y. Usuki, S. Takeda, N. Shiratori, Y. Katod, and J. Kimb, "A New Analytical Model of TCP Hybla for Satellite IP Networks," *Journal of Network and Computer Applications*, vol. 124, Dec. 2018.

[8] M. Dong, Q. Li, D. Zarchy, P. B. Godfrey, and M. Schapira, "PCC: Re-architecting Congestion Control for Consistent High Performance," in *Proceedings of the 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Oakland, CA, USA, 2015.

[9] S. Claypool, J. Chung, and M. Claypool, "Comparison of TCP Congestion Control Performance over a Satellite Network," in *Proceedings of the Passive and Active Measurement Conference (PAM)*, Virtual Conference, Apr. 2021.

[10] N. Cardwell and Y. Cheng and C. S. Gunn and S. H. Yeganeh and Van Jacobson, "BBR: Congestion-based Congestion Control," *Communications of the ACM*, vol. 60, no. 2, pp. 58–66, Jan. 2017.

[11] C. Caini and R. Firrincieli, "TCP Hybla: a TCP Enhancement for Heterogeneous Networks," *International Journal of Satellite Communications and Networking*, vol. 22, no. 5, pp. 547–566, Sep. 2004.

[12] S. Floyd, J. Mahdavi, M. Mathis, and A. Romanow, "TCP Selective Acknowledgment Options," *IETF Request for Comments (RFC) 2018*, Oct. 1996.

[19] M. Dong, T. Meng, D. Zarchy, E. Arslan, Y. Gilad, B. Godfrey, and M. Schapira, "PCC Vivace: Online-Learning Congestion Control," in *Proceedings of the 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, Renton, WA, USA, Apr. 2018.

[13] Y. Cao, A. Jain, K. Sharma, A. Balasubramanian, and A. Gandhi, "When to Use and When not to Use BBR: An Empirical Analysis and Evaluation Study," in *Proceedings of the Internet Measurement Conference (IMC)*, Amsterdam, NL, Oct. 2019.

[14] R. Ware, M. K. Mukerjee, S. Seshan, and J. Sherry, "Modeling BBR's Interactions with Loss-Based Congestion Control," in *Proceedings of the Internet Measurement Conference (IMC)*, Amsterdam, Netherlands, Oct. 2019.

[15] B. Turkovic, F. A. Kuipers, and S. Uhlig, "Interactions Between Congestion Control Algorithms," in *Proceedings of the Network Traffic Measurement and Analysis Conference (TMA)*, Paris, France, 2019.

[16] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: New Techniques for Congestion Detection and Avoidance," *ACM SIGCOMM Computer Communication Review*, vol. 24, no. 4, 1994.

[17] A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, , and F. Yang, "The QUIC Transport Protocol: Design and Internet-Scale Deployment," in *Proceedings of the ACM SIGCOMM Conference*, Los Angeles, CA, USA, Aug. 2017.

[18] Y. Wang, K. Zhao, W. Li, J. Fraire, Z. Sun, and Y. Fang, "Performance Evaluation of QUIC with BBR in Satellite Internet," in *Proceedings of the 6th IEEE International Conference on Wireless for Space and Extreme Environments (WiSEE)*, Huntsville, AL, USA, Dec. 2018.

[20] S. Claypool, J. Chung, and M. Claypool, "Measurements Comparing TCP Cubic and TCP BBR over a Satellite Network," in *Proceedings of the IEEE Consumer Communications & Networking Conference (CCNC)*, Virtual Conference, Jan. 2021.

[21] S. Floyd, "Metrics for the Evaluation of Congestion Control Mechanisms," Internet Requests for Comments, RFC 5166, March 2008.

[22] N. Cardwell, Y. Cheng, S. H. Yeganeh, and V. Jacobson, "BBR Congestion Control," *IETF Draft draft-cardwell-iccrg-bbr-congestion-control-00*, Jul. 2017.

[23] S. Claypool, M. Claypool, J. Chung, and F. Li, "Sharing but not Caring - Performance of TCP BBR and TCP CUBIC at the Network Bottleneck," in *Proceedings of the 15th IARIA Advanced International Conference on Telecommunications (AICT)*, Nice, France, Aug. 2019.

[24] B. Peters, P. Zhao, J. W. Chung, and M. Claypool, "TCP HyStart Performance over a Satellite Network," in *Proceedings of the 0x15 NetDev Conference*, Virtual Conference, Jul. 2021.

[25] S. Ha and I. Rhee, "Hybrid Slow Start for High-Bandwidth and Long-Distance Networks," in *International Workshop on Protocols for Fast Long-Distance Networks*, Manchester, UK, Mar. 2008.

[26] R. Jain, *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John Wiley and Sons, Inc., 1991.

[27] L. Guo, Y. Liu, W. Yang, Y. Zhang, and J. Lee, "Stateful-BBR – An Enhanced TCP for Emerging High-Bandwidth Mobile Networks," in *Proceedings of the IEEE/ACM 29th International Workshop on Quality of Service*, Virtual Conference, Jun. 2021.