The Effects of a Performance Enhancing Proxy on TCP Congestion Control over a Satellite Network

Mingxi Liu[*], Yongcheng Liu[*], Zhifei Ma[*], Zachary Porter[*], Jae Chung[†],
Saahil Claypool[*], Feng Li[†], Jacob Tutlis[*], Mark Claypool[*]
[*] Worcester Polytechnic Institute, Worcester, MA, USA
email: mliu4, yliu31, zma4, zaporter, smclaypool, jtutlis, claypool @wpi.edu
[†] Viasat, Marlboro, MA, USA
email: jaewon.chung, feng.li @viasat.com

***Abstract*—Satellite networks often use Performance Enhancing Proxies (PEPs) in order to overcome the inherent high latencies that are detrimental to TCP throughputs. Measurements of TCP performance over Satellite PEPs are lacking, both for actual PEP benefits and for interactions between PEPs and TCP congestion control algorithms. This paper presents results from experiments that assess the benefits of a PEP for a commercial satellite network, considering four TCP congestion control algorithms: Cubic, BBR, Hybla and PCC. Without the PEP, the four algorithms have similar steady state throughputs (about 70 Mb/s), but significant differences in start-up throughputs. In particular, the PEP dramatically improves (3x) start-up throughputs for TCP Cubic – the default congestion control algorithm used by most Internet servers. Overall, the PEP equalizes performance irrespective of the TCP congestion control algorithm chosen.**

***Index Terms*—PEP, throughput, round-trip time, retransmission**

I. Introduction

Satellite networks are an essential part of modern network infrastructures, providing ubiquitous network connectivity even in times of disaster. The number of satellites in orbit is over 2100, a 67% increase from 2014 to 2019 [2]. As few as three geosynchronous (GSO) satellites can provide global coverage, interconnecting widely distributed networks and providing "last mile" connectivity to remote hosts. The idea of "always on" connectivity is particularly useful for redundancy, especially in an emergency when traditional (i.e., wired) connections may not be available. Recent research in satellite technology has produced spot beam frequency reuse to increase transmission capacities more than 20-fold, and the total capacity of planned GSO satellites is over 5 Tb/s.

While throughput gains for satellite Internet are promising, satellite latencies remain an issue. GSO satellites orbit about 22k miles above the earth, so the physics for communication between terrestrial hosts using a satellite means about a 550 millisecond round-trip time at a minimum [8] – a challenge for TCP-based protocols that rely upon round-trip time communication to advance their data windows. To mitigate these latencies that can limit TCP throughput, many satellite operators use middle-boxes, typically known as "performance enhancing proxies" (or PEPs), to short-circuit the round-trip time communication to the satellite. As depicted in Figure 1, with a satellite PEP, the end-to-end TCP connection is split into 3 separate TCP connections, with the leg that traverses the satellite connection (TCP 2) tuned for high latency. The end system hosts use standard, unmodified TCP (TCP 1 and TCP 3) and do not necessarily even know that a PEP has split the connection. Without the PEP, the end hosts (servers and client) communicate as normal, with a single, unbroken TCP connection.

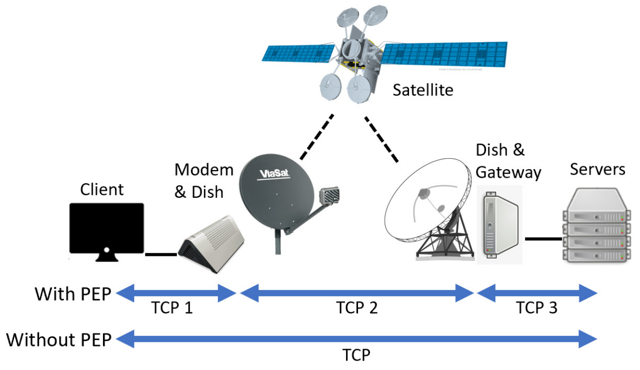


Fig. 1: TCP over a satellite with and without a performance enhancing proxy.

In addition to the presence (or absence) of a PEP, TCP congestion control algorithms play a critical role in enhancing or restricting throughput in the presence of network loss and latency. TCP Cubic [16] is the default TCP congestion control algorithm in Linux and Microsoft Windows, but BBR [19] has been widely deployed by Google on Linux servers and is a congestion control option available in the QUIC transport protocol [6], as well. A better understanding of TCP congestion control algorithm performance over satellite networks with and without a PEP is needed in order assess challenges and opportunities that satellites have to better support TCP moving forward.

However, while the types and prevalence of PEPs on the Internet has been explored [26], as has PEP performance for mobile networks [17], [27], satellite PEP studies have been limited to Web caches only [14]. TCP Cubic and TCP BBR measurements have been done over wireless networks [18], but there are few published studies measuring network performance over actual satellite networks [20], with most studies either using just simulation [3] or emulation with satellite parameters [1], [12], [24], [25], or with a satellite but without assessing a PEP [9], [23].

This paper presents results from experiments that measure the impact of a performance enhancing proxy in a commercial satellite Internet network, considering the interaction effects with the TCP congestion control algorithm. We compare the effects of a PEP for four TCP algorithms with different approaches to congestion control: default loss-based (Cubic [16]), bandwidth estimation-based (BBR [19]), utility function-based (PCC [12]), and satellite-optimized for start-up (Hybla [4]). The network testbed and experiments are done over the Internet for ecological validity, but designed to be as comparable across runs as possible by interlacing runs of each protocol serially to minimize temporal differences and by doing 80 bulk download runs over 24 hours for each protocol to provide for a large sample.

Analysis of the results shows the PEP improves performance on average for all protocols, with rather small effects on steady state throughput but pronounced benefits to TCP start-up. In particular, TCP Cubic, the default congestion control algorithm for most Internet servers, gets about a 3x boost to start-up performance from the PEP with a more modest (less than 1%) boost to steady state throughput. With the PEP, performance is about the same regardless of the type of TCP congestion control algorithm.

The rest of this report is organized as follows: Section II gives an overview of 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 performance enhancing proxies (PEPs) (Section II-A), TCP congestion control algorithms (Section II-B), and TCP performance over satellite networks (Section II-C).

A. *Performance Enhancing Proxies*

Ehsan et al. [14] evaluate the benefits to performance for a satellite Internet network using a Web cache PEP based on log analysis. They show performances gains from the PEP are sensitive to the amount of packet loss but do not explore benefits of the PEP for high latency.

Ivanovich et al. [17] present results from a PEP that seeks to mitigate the negative effects of wireless packet loss on TCP performance. Results simulating their proxy over 3G networks shows some improvements to TCP throughputs for 1%, 2% and 3% packet error rates.

Xu et al. [27] assess the behavior of transparent Web proxies (a type of PEP) in U.S. mobile networks, describing different types of PEPs based on how they handle Web content. They find split connection proxies, such as those studied in our paper, do not necessarily enhance performance, instead only having noticeable benefits with large flows and/or high link loss.

Weaver et al. [26] detect PEPs based on Netalyzr logs, identifying proxy locations and their intended use. They find 14% of Netalyzer-analyzed clients hint at the presence of Web proxies. They do not, however, assess the effects of the PEPs on performance.

Pavur et al. [22] design and implement QPEP, an open-source PEP/VPN hybrid based on the QUIC standard that has the potential to encrypt all traffic over a satellite link. Their experiments show QPEP provides up to 72% faster page load times compared to traditional VPN encryption.

While helpful to understand the scope of PEP features and use and effects PEPs have with packet loss, the above papers do not focus on mitigating the effects of latency nor do they explore any interactions with TCP congestion control algorithms – in fact, some of the congestion control algorithms we study did not exist at the time of this previous work. Our paper seeks to start remedying these shortcomings.

B. *TCP Congestion Control (CC)*

There have been numerous proposals for improvements to TCP's congestion control algorithm since its inception. Those most relevant to our work are highlighted here, in chronological order.

Caini and Firrinielli [4] propose TCP Hybla to overcome the limitations TCP NewReno flows have when running over high latency links (e.g., a Satellite). TCP Hybla modifies the standard congestion window increase with an extension based on the round-trip time. In Hybla slow start, $cwnd = cwnd + 2^{\rho} - 1$ and in congestion avoidance, $cwnd = cwnd + \frac{\rho^2}{cwnd}$, where $\rho = RTT/RTT_0$. RTT_0 is 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. [16] 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. TCP Cubic has been the default in Linux as of kernel 2.6.19 (in 2007), Windows 10.1709 Fall Creators Update (in 2017), and Windows Server 2016 1709 update (in 2017).

Dong et al. [12] propose TCP PCC that observes performance based on small measurement "experiments". The experiments assess throughput, loss, and round-trip times with a utility function, adopting the rate that has the best utility. PCC is not generally available for Linux, but Compira Labs[1] provided us with a Linux-based implementation.

Cardwell et al. [19] provide TCP Bottleneck Bandwidth and Round-trip time (BBR) as an alternative to Cubic's (and Hybla's) loss-based congestion control. BBR uses the maximum bandwidth and minimum round-trip time observed to 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 as of kernel 4.9 (end of 2016).

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

C. *TCP over Satellite Networks*

Obata et al. [20] evaluate TCP performance over actual (not emulated, as is typical) satellite networks. They compare a satellite-oriented TCP congestion control algorithm (STAR) with NewReno and Hybla. Experiments with the Wideband InterNetworking Engineering test and Demonstration Satellite (WINDS) network 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 link than TCP NewReno. We evaluate TCP Hybla, but there is no public Linux implementation of TCP STAR available.

Wang et al. [25] provide preliminary performance evaluation of QUIC with BBR on an emulated 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 throughput improvements compared with Cubic for their emulated satellite network.

Utsumi et al. [24] develop an analytic model for TCP Hybla for steady state throughput and round-trip time 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 TCP Hybla has substantial improvements to throughput over that of TCP Reno for loss rates above 0.0001%

Cao et al. [5] analyze measurement results of TCP BBR and TCP Cubic over a range of different network conditions. They produce heat maps and a decision tree that identifies conditions which show performance benefits for using 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.

Claypool et al. [9] provide comparative performance for TCP congestion control algorithms on a commercial satellite Internet network. Analysis shows similar steady state bitrates for all algorithms, but with significant differences in start-up throughputs and round-trip times caused by queuing of packets in flight. Unlike our paper, all their experiments are *without* a PEP.

Our current work extends the above work by using a commercial satellite network, not simulation, considering four TCP congestion control algorithms, not one or two, and directly assessing the impact of a split-connection, performance enhancing proxy (PEP).

III. Methodology

To evaluate TCP congestion control over a satellite link with and without a performance enhancing proxy (PEP), we setup a testbed (Section III-A), serially bulk-download data using different TCP congestion control algorithms with the PEP either on or off (Section III-B), and analyze the results (Section IV).

A. *Testbed*

We setup a Viasat[2] satellite Internet link so as to represent download to a client with a "last mile" satellite connection. Our servers are configured to allow for repeated tests and comparative performance by using consecutive serial runs with all conditions the same except for: 1) the PEP on/off, and 2) the 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 four servers, each with a different TCP congestion control algorithm: Cubic, BBR, Hybla and 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 our University LAN via Gb/s Ethernet. The campus network is connected to the Internet via several 10 Gb/s links, all throttled to 1 Gb/s. Wireshark captures all packet header data on each server and the client. The client connects to a Viasat satellite terminal (with a dish and modem) via a Gb/s Ethernet connection. The client's downstream Viasat service plan provides a peak data rate of 144 Mb/s.

The terminal communicates through a Ka-band outdoor antenna (RF amplifier, up/down converter, reflector and feed) through the Viasat 2 satellite[3] 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 Viasat gateway performs per-client queue management, where the queue for each client can grow up to 36 MBytes, allowing a maximum queuing delay of about 2 seconds at the peak data rate. Queue lengths are controlled at the gateway by Active Queue Management (AQM) that randomly drops 25% of incoming packets when the queue is over half of the limit (i.e., 18 MBytes).

Previous work [9] assessed link baseline (i.e., without any induced traffic) performance and shows the vast majority (99%) of round-trip times are from 560 and 625 milliseconds, and average loss rates are about 0.05%, with most of these (77%) single-packet losses.

Given the 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.

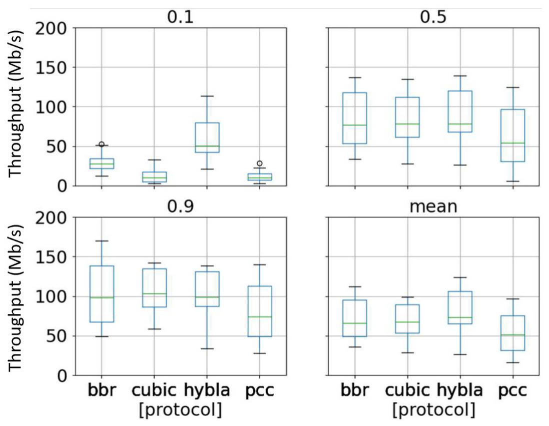
The PEP (the Viasat Web Accelerator) can be enabled (on) or disabled (off) at the terminal. When enabled, TCP flows from server to client are transparently (to the client and server) split into 3 legs – server to gateway (TCP 3), gateway to terminal (TCP 2), and terminal to client (TCP 1). In contrast, when the PEP is disabled (off), there is a single TCP flow from the server to the client.

B. *Downloads*

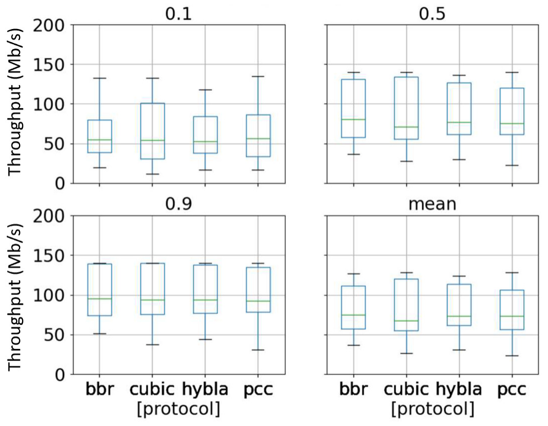
We compare the performance of four congestion control algorithms, chosen as representatives of different congestion control approaches: loss-based Cubic; bandwidth-delay product-based BBR (version 1); utility function-based PCC;

[2]https://www.viasat.com/internet

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



(a) Performance enhancing proxy off.



(b) Performance enhancing proxy on.

Fig. 2: Overall throughput distributions for 10%, 50%, 90% and mean.

and satellite-optimized, loss-based Hybla. The four servers are configured to provide for bulk-downloads via `iperf3`[4] (v3.3.1), each server hosting one of our four congestion control algorithms. Cubic, BBR and Hybla are used without further configuration. PCC is configured to use the Vivace-Latency utility function [13], with throughput, loss, and round-trip time coefficients set to 1, 10, and 2, respectively.

For all hosts, the default TCP buffer settings are changed on both the server and client – setting `tcp_mem`, `tcp_wmem` and `tcp_rmem` to 60 MBytes – so that the downloads are not flow-controlled and instead are governed by TCP's congestion window [23].

The client turns the PEP on, initiates a connection to one server via iperf, downloads 1 GByte, then does a repeat but with the PEP off. After the two downloads (one with the PEP on, one with the PEP off), the client proceeds to the next server (i.e., to another congestion control algorithm). After cycling through each of the four servers, the client pauses for 1 minute. The process repeats a total of 40 times – thus, providing 80 network traces of a 1 GByte download for each protocol over the satellite link, half with the PEP on and half with the PEP off. Since each cycle takes about 15 minutes, the throughput tests run for about a day total. We analyze results for a weekday.

IV. Analysis

A. Overall

The throughput for each download is computed from the Wireshark traces on the downlink from the server. Figure 2 depicts the overall throughput boxplot distributions at different percentiles taken across all flows and grouped by protocol. PEP off results are in Figure 3a and PEP on results in Figure 3b. For both graphs, the top left is the tenth percentile, the top right the 50% (or median), the bottom left the ninetieth percentile and the bottom right the mean. Each box depicts quartiles and median for the distribution. Points higher or lower than 1.4 $\times$ the inter-quartile range are outliers, depicted by the circles. The whiskers span from the minimum non-outlier to the maximum non-outlier.

For the PEP on versus the PEP off, the biggest visual difference is in the tenth percentile with the PEP-on throughputs being near 50 Mb/s while the PEP-off throughputs are 1/4 to 1/2 as much, except for Hybla which appears similar in both. For the median, ninetieth percentile and mean, the PEP-on/off results look similar. When the PEP is on, all protocols have visually similar throughput distributions at each percentile and for the mean. When the PEP is off, Cubic, BBR and Hybla have visually similar throughput distributions at the median, ninetieth percentile and mean, while PCC has lower throughput distributions in all cases.

B. Steady State

TCP's overall performance includes both start-up and congestion avoidance phases – the latter we call "steady state". We analyze steady state behavior based on the last half (in terms of bytes) of each trace.

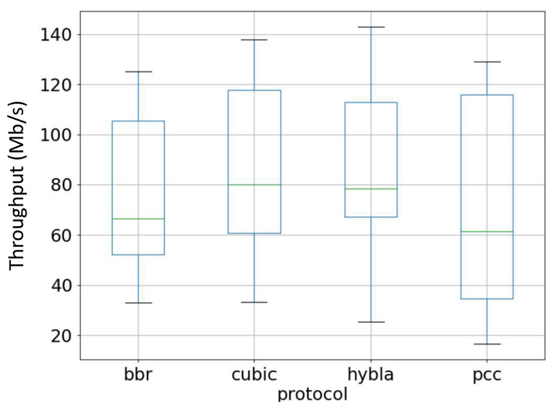
Figure 3 shows the boxplot distributions for the mean throughputs for each protocol, PEP-off in Figure 3a and PEP-on in Figure 3b. The boxes and whiskers are the same as in Figure 3. Comparing Figure 3a to Figure 3b, the steady state distributions with the PEP off are similar to those with the PEP on, with the latter slightly shifted upwards – i.e., q1 and q3 are higher with the PEP on for all protocols. The median is similarly higher for all, with the exception of Cubic.

C. Start-Up

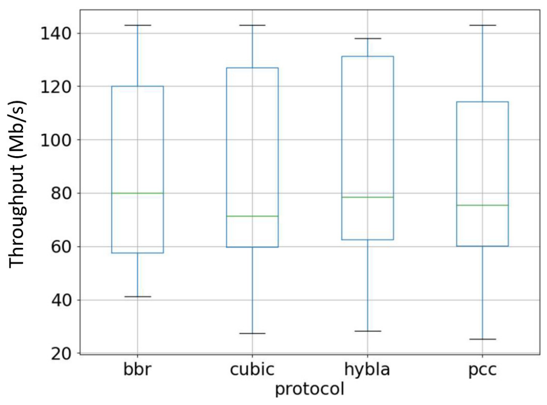
We compare the start-up behavior for each protocol by analyzing the first 30 seconds of each trace, approximately long enough to download 50 MBytes on our satellite link. This is indicative of algorithm performance for some short-lived flows and is about when we observed throughput growth over time "flattening" for most flows.

The average Web page size for the top 1000 sites was around 2 MBytes as of 2018 [11], including HTML payloads and all

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

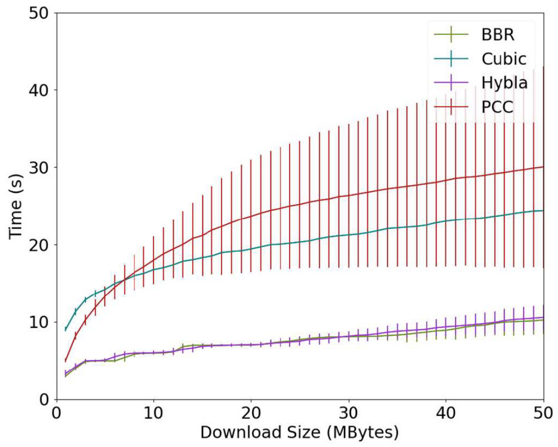


(a) Performance enhancing proxy off.

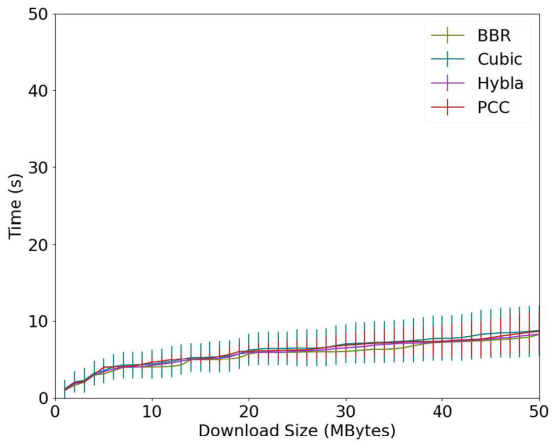


(b) Performance enhancing proxy on.

Fig. 3: Steady state mean throughput distributions.



(a) Performance enhancing proxy off.



(b) Performance enhancing proxy on.

Fig. 4: Download time versus download object size.

linked resources (e.g., CSS files and images). The Web page size distribution's 95th percentile was about 6 MBytes and the maximum was about 29 MBytes. Today's average total Web page size is probably about 5 MBytes [15], dominated by images and video.

Many TCP flows stream video content and the amount streamed depends upon the video encoding. However, assuming videos are downloaded completely, about 90% of YouTube videos are less than 30 MBytes [7].

Figure 4 depicts the time on the y-axis (in seconds) to download an object for the given size on the x-axis (in MBytes), PEP-off in Figure 4a and PEP-on in Figure 4b. The object size increment is 1 MByte. Each point is the mean time required by a protocol to download an object of the indicated size, shown with a 95% confidence interval.

Comparing PEP-off (Figure 4a) to PEP-on (Figure 4b), the PEP has a dramatic benefit to download times for PCC and Cubic, decreasing times by about 3x, but with less impact on BBR and Hybla. In fact, with the PEP on, the start-up performance for all protocols is about the same. With the PEP off, for the smallest objects (1 MByte), Hybla and BBR download the fastest, about 4 seconds, PCC about 6 seconds and Cubic about 9.5 seconds. For an average Web page download (5 MBytes), Hybla and BBR take an average of about 5 seconds, PCC 11 seconds, and Cubic 13 seconds. For 90% of all videos and the largest Web pages (30 MBytes), Hybla and BBR take about 8 seconds, Cubic about twice that and PCC about thrice. With the PEP on, for all protocols, the average Web page download takes about 5 seconds and 90% of all video downloads and the largest Web page downloads take about 8 seconds.

D. Summary

Table I shows summary statistics (mean μ, standard deviation s) for the throughput *differences* for the PEP on and the PEP off. Positive values indicate the protocol had a higher bitrate with the PEP on. The table also shows the computed effect sizes. An effect size provides a measure of the magnitude of difference – in our case, the difference of the means with the PEP on versus the PEP off. In short, the effect size quantifies how much having the PEP on matters. The Glass' Δ effect size assesses the differences in means in relation to the standard deviation of the control group (here, the PEP off). Generally small effect sizes are anything under 0.2, medium are 0.2 to 0.5, large (shown with *italics*) 0.5 to 0.8, and **very large** (shown with **bold**) above 0.8.

TABLE I: Throughput summary statistics (in Mb/s) for mean (μ), standard deviation (s) and effect size (Δ). Values are differences with the PEP on and the PEP off – positive values indicate the protocol had a higher bitrate with the PEP on.

	Overall			Steady State			Start-up		
	μ	s	Δ	μ	s	Δ	μ	s	Δ
Cubic	10.8	16.4	*0.7*	0.8	17.5	0.0	24.6	7.4	**3.4**
BBR	10.1	9.5	**1.0**	11.0	13.8	*0.8*	7.4	6.5	**1.1**
Hybla	-1.0	9.6	0.1	-2.6	20.0	0.1	5.1	4.5	**1.0**
PCC	27.6	25.5	**1.4**	12.9	31.0	0.4	26.2	6.6	**4.3**

From the table, overall, the PEP provides the greatest benefit to PCC with a large effect. The throughput benefit to BBR is about the same as that to Cubic, but the effect is larger for BBR owing to BBR's lower standard deviation. The PEP has little effect on Hybla. At steady state, the effects of the PEP are smaller for most protocols, save BBR which benefits with a medium effect – likely because the PEP avoids the low bitrates caused when BBR probes for the minimum round-trip time for several seconds. At start-up, the benefits of the PEP are very large for all protocols, particularly so for Cubic and PCC.

E. Comparison to Related Work

We provide a high-level comparison of our results to some previously published results. Since a direct comparison is not possible as the studies cover different network conditions and protocol versions – one of the merits of our work is results for a previously untested set of network conditions – we examine the closest experimental conditions reported in each paper to those in our work: A) network link with about a 600 ms round-trip time, 140 Mb/s capacity, minimal loss, and a bottleneck queue of about 2x BDP, and B) comparisons of congestion control algorithm to Cubic. Since the benefits for a split connection PEP are primarily in reducing the round-trip time during slow start, we examine the throughput during start-up.

Table II depicts the results, broken into two parts: 1) *PEP comparison*, showing the gains with PEP on versus PEP off, and 2) *Protocol comparison*, showing the gains with alternative protocols to Cubic – here, with the PEP off. "Capacity" and "RTT" (round-trip time) are as specified in the paper and/or based on measured values. "Gain" is the maximum throughput benefit obtained from either the PEP or the alternate protocol. "Notes" indicates key differences in the experiments. Previous work indicates the bottleneck buffer size matters for BBR [5], [10] and PCC [13] performance, so it is indicated where applicable. For our work, the buffer size is about 1 BDP.

From the table, the gains in our experiments both from the PEP and from Hybla and BBR are generally larger than those reported in other works. This may be due to the larger capacity on our satellite Internet link and the accompanying with high round-trip times making the BDP larger than studied in previous work. The notable exception to this gain is for PCC, where the previous 20x gains from PCC compared to our 1.2x gains are likely due to their small buffer size at the bottleneck.

TABLE II: Performance Comparison Summary

Paper	Capacity	RTT	Gain	Notes
		PEP Comparison		
		PEP on vs. PEP off		
Ehsan et al. [14]	24 Mb/s	500 ms	**0.75x**	Satellite
Xu et al. [27]	1 Mb/s	200 ms	**2x**	Mobile
Ours	140 Mb/s	600 ms	**3x**	Gain depends on protocol
		Protocol Comparison		
		BBR vs. Cubic		
Wang et al. [25]	1	600	**0.05x**	Emulation, Only QUIC
Cao et al. [5]	100	200	**0.1x**	Emulation, 4 BDP buffer
Ours	140	600	**3x**	
		Hybla vs. Cubic		
Caini et al. [4]	10	600	**2x**	Emulation, vs. NewReno
Obata et al. [20]	42	800	**0.1x**	Satellite, vs. NewReno
Ours	140	600	**3x**	
		PCC vs. Cubic		
Dong et al [13]	42	800	**20x**	Emulation, 0.25 BDP buffer
Ours	140	600	**1.2x**	

V. Conclusion

While satellite Internet connections are important for providing reliable connectivity, to date, there are few published research papers measuring the benefits of a PEP in an actual satellite network, in particular in combination with different TCP congestion control algorithms. This paper presents results from experiments on a production satellite network, comparing the effects of a PEP on TCP considering four congestion control algorithms – the two dominant algorithms, Cubic and BBR, a commercial implementation of PCC, and the satellite-tuned Hybla.

For overall throughput, the PEP benefits PCC the most owing to PCC's relatively low throughput with high link round-trip times. The PEP has a large effect on Cubic and BBR, improving average throughput by about 10 Mb/s for each. The proxy has little effect on Hybla which is designed for high round-trip times without a PEP. During steady state, the PEP has much less benefit to all protocols since they benefit little once congestion windows have grown large enough to saturate the high round-trip time link. However, during start-up, the benefits of the PEP are at least large for all protocols, and very large for Cubic and PCC that are slow to ramp up to the link capacity with the high link round-trip times. Moreover, with the proxy, start-up performance is similar for all protocols, with small downloads (e.g., Web pages) taking about 5 seconds, about 3 times faster than a default Cubic flow without a PEP.

The results pertain to geosynchronous satellite networks, but should apply to other networks with high bandwidth-delay products. The results may also be relevant to other satellite networks, such as low earth orbit satellites [21], as these have relatively lower latencies (about 50 ms) but potentially higher bandwidth.

There are several areas we are keen to pursue as future work. Settings to TCP, such as the initial congestion window, may have a significant impact on performance, especially for small object downloads [23]. Video streams using HTTP are also of interest given their TCP dynamics and prevalence on many network links. Since prior work has shown TCP BBR does not always share a bottleneck network connection equitably with TCP Cubic [10], future work is to run multiple flow combinations over the satellite link with the PEP on and off. Since encrypted flows cannot be split by a PEP, future work is to evaluate the performance of implementations of QUIC [6] on the satellite link.

Acknowledgments

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

References

[1] V. Arun and H. Balakrishnan, "Copa: Practical Delay-Based Congestion Control for the Internet," in *Proceedings of the Applied Networking Research Workshop*, Montreal, QC, Canada, Jul. 2018.

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

[3] 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.

[4] 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.

[5] 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.

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

[7] X. Che, B. Ip, and L. Lin, "A Survey of Current YouTube Video Characteristics," *IEEE Multimedia*, vol. 22, no. 2, April - June 2015.

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

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

[10] 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.

[11] Data and Analysis, "Webpages Are Getting Larger Every Year, and Here's Why it Matters," Solar Winds Pingdom. Online at: https://tinyurl.com/y4pjrvhl, November 15 2018.

[12] 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.

[13] 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.

[14] N. Ehsan, M. Liu, and R. Ragland, "Evaluation of Performance Enhancing Proxies in Internet over Satellite," *International Journal of Communication Systems*, vol. 16, no. 6, 2003.

[15] T. Everts, "The Average Web Page is 3 MB. How Much Should We Care?" Speed Matters Blog. Online at: https://speedcurve.com/blog/web-performance-page-bloat/, August 9th 2017.

[16] 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.

[17] M. Ivanovich, P. Bickerdike, and J. Li, "On TCP Performance Enhancing Proxies in a Wireless Environment," *IEEE Communications Magazine*, vol. 46, no. 9, Sep. 2008.

[18] F. Li, J. W. Chung, X. Jiang, and M. Claypool, "TCP CUBIC versus BBR on the Highway," in *Proceedings of the PAM Conference*, Berlin, Germany, Mar. 2018.

[19] 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.

[20] 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.

[21] C. H. Park, P. Austria, Y. Kim, and J.-Y. Jo, "MPTCP Performance Simulation in Multiple LEO Satellite Environment," in *IEEE Computing and Communication Workshop and Conference (CCWC)*, Virtual Conference, Jan. 2022.

[22] J. Pavur, M. Strohmeier, V. Lenders, and I. Martinovic, "QPEP: An Actionable Approach to Secure and Performant Broadband from Geostationary Orbit," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, Feb. 2021.

[23] 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.

[24] 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.

[25] 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.

[26] N. Weaver, C. Kreibich, M. Dam, and V. Paxson, "Here Be Web Proxies," in *Proceedings of PAM*, Los Angeles, CA, USA, Mar. 2014.

[27] X. Xu, Y. Jiang, T. Flach, E. Katz-Bassett, D. Choffnes, and R. Govindan, "Investigating Transparent Web Proxies in Cellular Networks," in *Proceedings of the PAM Conference*, New York, NY, USA, Mar. 2015.