Cascaded TCP: Applying Pipelining to TCP for Efficient Communication over Wide-Area Networks

Umar Kalim*†, Mark K. Gardner†, Eric Brown†, Wu-chun Feng*
*Department of Computer Science, †Office of IT, Virginia Tech
umar@cs.vt.edu, {mkg,eric.brown}@vt.edu, feng@cs.vt.edu

***Abstract*—The bandwidth utilization in traditional TCP protocols (e.g., TCP New Reno) suffers over high-latency and high-bandwidth links due to the inherent characteristics of TCP congestion control. Conventional methods of improving throughput cannot be applied per se for streaming applications. The challenge is exacerbated by "big data" applications such as with the *Long Wavelength Array* data that is generated at a rate of up to 4 terabytes per hour.**

To improve bandwidth utilization, we introduce layer-4 relay(s) that enable the pipelining of TCP connections. That is, a traditional end-to-end connection is split into independent streams, each with shorter latencies, that are then concatenated (or cascaded) together to form an equivalent end-to-end TCP connection. This addresses the root cause by decreasing the latency over which the congestion-control protocol operates.

To understand when relays are beneficial, we present an analytical model, empirical data and its analyses, to validate our argument and to characterize the impact of latency and available bandwidth on throughput. We also provide insight into how relays may be setup to achieve better bandwidth utilization.

***Index Terms*—bandwidth utilization; throughput; performance; long-fat networks; pipelining; TCP; wide-area networks**

I. Introduction

A collaboration between Virginia Tech and University of New Mexico requires the transfer of streaming data from the Long Wavelength Array (LWA) in New Mexico to Blacksburg, Virginia. LWA is capable of generating $4\ TB$ of data per hour, and data rates will grow by $53X$ as the instrument roles out. Conventional methods of optimizing network bandwidth usage, which focus on file transfers, do not apply per se to such streaming data.

To exacerbate the problem, TCP New Reno [1], the most widely deployed TCP congestion-control algorithm, delivers poor average throughput over paths with high bandwidth-delay products. This results in poor bandwidth utilization. The algorithm is designed such that the sender has to wait for the receiver's feedback before the window sizes may be updated, and therefore TCP New Reno cannot react fast enough over high-latency links to achieve higher throughput.

We provide guidance for the deployment of TCP streaming relays so as to reduce the impact of large end-to-end latencies. As illustrated in Figure 1, a relay takes responsibility for pipelining traffic from the source to the destination. The connection from the source is terminated at the relay; the relay in turn creates an independent (or cascaded) TCP connection to forward data towards the destination. Consequently, a *divide-and-conquer* approach allows the sliding-window protocols operating on either side of each relay to receive control feedback much faster than the *long-haul* connection, thus delivering better bandwidth utilization. We refer to such a setup as *Cascaded TCP* [2]. Note that such an approach would benefit both streaming data as well as typical file transfer.

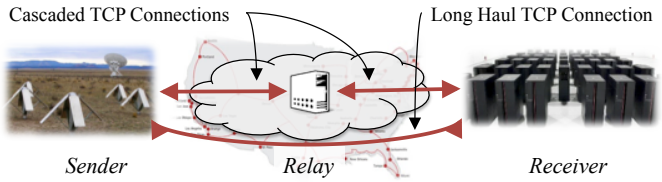


Fig. 1. The long-haul (end-to-end) TCP connection is split into two independent TCP connections by the relay, each with smaller latencies.

Reducing the adverse impact of large latencies on throughput by using relays is not a novel idea. However the use of relays is not appropriate for all scenarios. Thus, there is a need to determine when Cascaded TCP may be applied.

Our contributions in this paper are:

- An analytical model for Cascaded TCP that provides guidance for when to use relays.
- An evaluation of the use of TCP relays to improve aggregate throughput and to test the hypothesis that the use of relays improves bandwidth utilization.

We validate our hypothesis via an empirical study.

We present the related work in Section II and describe the analytical model for Cascaded TCP in Section III. Section IV describes the testbed and experiments. Section V explains the empirical results. We evaluate the analytical model and results in Section VI. Conclusions and directions of further research are presented in Section VIII.

II. Related Work

As Cheshire suggested, with poor latency there is only so much that one can do [3]. Nevertheless, by using a divide-and-conquer approach, we can address the challenges imposed by large latencies. Border et al. make a case for performance-enhancing proxies (PEPs) [4], where they propose a generic divide-and-conquer approach to improve performance. They argue that PEPs may be applied at the application, transport and/or link layer for the purposes of tunneling, compression, reliability or improving throughput. They suggest that the decision whether to use proxies "should be under the control of the end user" [4]. We build on PEPs and propose an analytical

model that can be automated to decide whether Cascaded TCP, PEPs, or the like should be used or not.

Many application acceleration proxies are available today from different vendors and inspired by performance-enhancing proxies. They are typically marketed as front-end-optimization, dynamic-site-optimization, and WAN-acceleration solutions.

Content distribution networks [5] use caches to house data near the users' location. While this reduces latency, it can only be applicable in scenarios where the data is available a priori. Such an approach may not be applied to "Big Data" applications, where the only option is to use the data or discard it, as in the case of the Long Wavelength Array.

Research on congestion-control to improve network performance has led to the development of algorithms such as BIC [6], CUBIC [7] and Compound TCP [8][1]. In contrast with TCP New Reno, a packet loss in these algorithms triggers an aggressive search for available bandwidth by making drastic changes to window sizes. While they do improve throughput, they adversely impact TCP friendliness and RTT fairness [9]. In contrast, Cascaded TCP reduces the latency that a connection experiences by segmenting the path.

On the other hand, algorithms such as XCP [10] and ECN [11] provide the sender with control information to avoid losses due to congestion. Subsequently low aggregate throughput can be avoided by minimizing losses and thereby side stepping the drastic changes in window sizes (i.e., reduction due to loss followed by recovery). However, XCP and ECN are effective only when appropriate active queue management policies are implemented. Cascaded TCP, on the other hand, reduces the latency for each TCP segment that forms the logical connection, thereby improving aggregate bandwidth, even with AQM.

Alternate methods of increasing throughput have been proposed to parallelize data transport. Research done by Sivakumar et al. [12] and Zhang et al. [13] are select examples. Part of the networking community is opposed to such methods because they are not TCP friendly. Note that Cascaded TCP can be combined with other optimizations; if throughput for a single flow can be improved with Cascaded TCP, the same may be applied to each flow of the parallel streams.

Sophisticated methods of communication such as hybrid (packet- and circuit-switched) networks have been analyzed by Veeraraghavan et al. [14] and use a decision process to choose between possible options. We seek to provide similar guidance but for packet-switched networks with relays.

Pucha and Hu [15] suggest overlay networking that enables layer-4 forwarding. Our research strengthens the argument for layer-4 forwarding as a means to increase bandwidth utilization with an analytical model that assists with the decision as to when Cascaded TCP should be used and with a comprehensive empirical analysis involving a wider spectrum of tests.

[1]BIC, CUBIC, and Compound TCP are not as widely deployed. FreeBSD and Mac OS X use TCP New Reno as the default congestion-control algorithm. Compound TCP, in Microsoft Windows Vista, is disabled by default. Linux kernel 2.6.19 and onwards use CUBIC as the default algorithm.

III. Analytical Model and Cascaded TCP

Let $T_{cascaded}$ be the time required to complete the transfer of data using a cascade of relays. We express this as

$$T_{cascaded} = T_{overhead} + T_{transfer} \tag{1}$$

where $T_{overhead}$ is the overhead of using cascaded TCP and $T_{transfer}$ is the time required to complete the transfer using relays. Note that $T_{overhead} = T_{setup} + T_{proc}$, where T_{setup} is the time required to setup a cascade of relays and associated TCP connections and T_{proc} is the processing overhead.

Given T_{lh} as the time required to transfer data using long-haul TCP, it would be prudent to choose Cascaded TCP if the overheads do not outweigh the benefits, i.e.,

$$T_{cascaded} < T_{lh} \quad \text{or} \tag{2}$$

$$T_{overhead} < T_{lh} - T_{transfer}. \tag{3}$$

A. Transfer Time for Long-Haul TCP (T_{lh})

We know that throughput is inversely proportional to latency between end-points. The rate of increase in throughput is also coupled with latency as the longer the latency, the longer it takes the congestion-control algorithm to increase the sender's TCP window size and converge towards the ideal bandwidth. The same conclusions can be derived from Mathis' model [16]:

$$throughput \leq \frac{cMSS}{RTT\sqrt{p}}, \tag{4}$$

where MSS = maximum segment size, RTT = latency, p = loss probability, $c = \sqrt{\frac{3}{2b}}$ such that $b = 1$ for long-haul TCP and $b = 2$ when delayed ACKs are enabled.

If S is the size of data, then using (4), the average transfer time over long-haul TCP is expressed as:

$$T_{lh} \leq \frac{S}{throughput} = \frac{S(RTT\sqrt{p})}{cMSS}. \tag{5}$$

There are much more precise models than Mathis' approximation such as that by Padhye et al. [17]. We chose Mathis' approximation for its simplicity, though Padhye's or other models may be used. However, the models cannot be applied to short-lived TCP flows as their entire lifetimes are usually within the slow-start phase.

B. Data Transfer Time for Cascaded TCP ($T_{transfer}$)

Cascaded TCP may be classified as non-pipelined or pipelined, based on the mechanics of the relay [2]. The relay is non-pipelined when it stores all the traffic coming from the sender until the connection is closed. Once the sender's connection closes, the relay starts forwarding traffic to the destination (or the next relay in the chain). It is understandable that such an approach would not be a viable option, particularly for streaming data. In contrast, a relay is pipelined when it is allowed to forward packets as soon as they are available in the queue. To avoid packet drought in the buffer — when a relay does not have enough data to send — the relay's outbound TCP connection may wait for W windows before

starting transmission. Therefore, transfer time $T_{transfer}$ or in particular the pipelined transfer time T_{pc} can be expressed as:

$$T_{transfer} = T_{pc} \leq \frac{S(RTT_k\sqrt{p_k})}{c\ MSS} + W\sum_{i=1}^{N} RTT_i, \quad (6)$$

where for each $i = 1, ..., N$, RTT_i denotes the round-trip time for TCP connection i. The first term is the bottleneck link transfer time. From (4), the bottleneck link would be $k = \arg\min\{i = 1, \ldots, N : RTT_i\sqrt{p_i}\}$ — that is the link with the lowest available bandwidth. The second term is the buffering to prevent drought and comes from the first W end-to-end round-trip times that each connection waits before starting its transmission. As window scaling is enabled for sizes beyond 64 KB and it takes 16 round trip times during the slow start phase for the window to grow beyond 64 KB, we choose 16 as a default value for W in (6).

Note that both (5) and (6) depend on latency (RTT) and loss (p). Here the message size may be considered a constant when choosing between long-haul and cascaded TCP.

C. Setup Time for Cascaded TCP (T_{setup})

The time to setup relays has a strong correlation with latency between source and relays. This is because setup time involves sending configuration parameters to the relay to trigger setup. Thus we can approximate setup time with the time it takes for the first payload segment to arrive. All relays along the path may be triggered in parallel. By doing so, the time to trigger relays that are closer to the sender is hidden by the time to trigger the relay that is the farthest. For the sake of simplicity we use the latency RTT between the sender and the receiver. A TCP connection is typically setup after one-and-a-half round trip. The first payload may arrive along with the third segment in the 3-way handshake. Therefore:

$$T_{setup} \approx 1.5\ RTT. \quad (7)$$

If this optimization were not applied and relays were setup in a sequence the overall setup time would be the sum of setup times for all the relays, that is $T_{setup} \approx \sum_{i=1}^{N} 1.5\ RTT_i$, where N is the number of relays.

D. Processing Overhead (T_{proc})

The processing overhead may be approximated by the time the relay process waits to avoid buffer drought, which is accounted for in (6). Note that the overhead of data passing through the relay's transport layer instead of being forwarded at layer 3 by a router would be negligible when compared to the waiting time to avoid buffer drought.

E. Summary

We can estimate T_{lh}, $T_{transfer}$, T_{setup} and T_{proc} from (5), (6) and (7), which allows us to estimate $T_{overhead}$ and evaluate the condition (3). We present and discuss the throughput estimates in relation to empirical measurements in Section VI-A.

IV. Experimental Setup

We use iperf v2.0.5 [18] to emulate the sender (client) and receiver (server) in our testbed. The nodes run FreeBSD 9.0. The relays are implemented with netcat [19] as layer-4 relays.

A. Approach

We configure the operating system appropriately (e.g., set kernel frequency timer at 4000 Hz, define maximum window sizes such that they do not gate bandwidth, enable window scaling and selective acknowledgements). We compute the expected bandwidth-delay product for the end-to-end path. This allows us to determine the transfer size for the combination of bandwidth capacity, end-to-end latency, and packet loss, to ensure that TCP connections remain in steady state for at least 90% of their lifetime. We then configure Dummynet pipes at the sender and relay(s) to emulate available bandwidth/capacity, end-to-end latency, and packet loss. We vary configurations for Dummynet pipes in the ranges listed in Table I. We measure achievable bandwidth, latency, and packet loss and take between 10 and 30 samples for each permutation of the parameters to compute statistical significance.

TABLE I
Values used to configure Dummynet and emulate testbed.

Metric	Range of Values
Round Trip Time (ms)	$8, 16, 32, 64, 128, 256, 512$
Bandwidth (Mbps)	$0.512, 1, 2, 4, 8, 16, 32, 64, 128, 256$
Packet Loss (%)	$0.001, 0.01, 0.1, 1$

1) Long-Haul (LH) TCP: The long-haul TCP connections provide a performance baseline for our testbed. To measure baseline performance, we route through the same path by enabling layer-3 forwarding at the relays. This is done to make the emulation overhead the same for both long-haul and Cascaded TCP. For long-haul tests, the sender is configured to have the first relay node as its gateway. The client addresses the server as the receiver/destination. Here the bandwidth estimates are computed as follows: $throughput = \frac{transfer\ size}{test\ duration}$.

2) Pipelined Cascaded TCP: Netcat is setup at the relays to act as layer-4 forwarding gateways. As data arrives at the relay, it is pipelined/forwarded to the next relay in the chain or to the destination if it is the last relay. In this case, we cannot assume that the test is complete when the client's TCP connection terminates; the relay may still be forwarding data when the sender's connection terminates. Therefore our measurements include the time until the last relay in the chain is done forwarding traffic to the server.

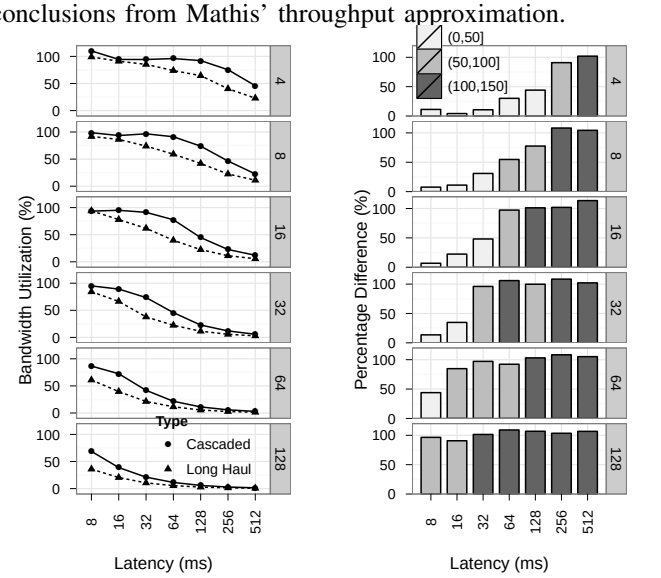
We recognize that Dummynet's emulation of bandwidth and latency is coarse-grained; Dummynet queues packets, which can increase delay in addition to what is configured. This coarse-grained emulation and bursty behavior is apparent in low-latency emulations where measured bandwidth results in observations greater than the limits defined.

V. Results

We collect and analyze results for all permutations of the metrics defined in Table I using one and two relays. Due to space constraints, we only present results for a loss factor of 0.1%, which we chose based on typical loss values [20].

A. Observations from the Testbed

1) Bandwidth utilization decreases with increase in latency: As summarized in Figure 2(a), we see that bandwidth utilization decreases with increased latency. We see similar trends for long-haul and Cascaded TCP. Utilization decreases further as loss increases. These observations remain valid for all varieties of bandwidth capacities we tested. We can derive the same conclusions from Mathis' throughput approximation.



(a) Bandwidth utilization

(b) Percentage increase of bandwidth utilization between Cascaded and long-haul TCP

Fig. 2. Bandwidth utilization with a single relay for varying link capacities (4 – 128 Mbps) and latency (8 – 512 ms) at a loss of 0.1%. The 95% confidence intervals are not shown here as all observations fall within ±5% of the mean.

For low-bandwidth limits and latencies, we observe that the bandwidth utilization is at times greater than 100%, which is not possible in reality. We attribute this to the coarse-grained emulation by Dummynet. This behavior is most apparent for low-latency configurations. We compute the transfer size based on the bandwidth-delay product such that the TCP connection remains in streaming state for about 90% of the time; for small bandwidth-delay products, the transfer sizes are small and tests complete in a few seconds. Variations in emulation at such granularities have significant impact on the results. In contrast, we do not see the same behavior for tests with large bandwidth-delay products as the errors are amortized over the duration of the test. We could have increased transfer sizes to amortize the effect of these errors. While doing so for short latencies amortized the errors, for large latencies the test durations became unreasonably long. An alternate approach would be to use different transfer size proportions for short and long latencies; in this case, however, we would not be doing a fair comparison across configurations.

We note that if loss and latency configurations remain the same and link capacities increase, the throughput achieved has a slight but noticeable decrease. This observation may be explained as follows. Assume a link with given bandwidth capacity and uniform loss. Here the window size will be able to grow to a certain percentage of the maximum size before loss is experienced and subsequently the window size drops. If the loss behavior remains the same, the percentage of maximum window size achieved is less for a high capacity link. This is observed in the empirical results.

2) Throughput improves by introducing layer-4 relay(s): Figure 2(a) shows that introducing a single relay results in significant improvement in bandwidth utilization due to better average throughput. By introducing a relay halfway, we split the connection into two TCP connections. Each split connection has shorter latency and therefore results in better aggregate throughput and thus better bandwidth utilization. Figure 2(b) provides a summary of the relative difference in bandwidth utilization. Cascaded TCP with a single relay delivers up to twofold improvement in bandwidth utilization.

Figure 2(a) shows that as latency increases, Cascaded TCP achieves increasing bandwidth utilization. The relative difference continues to grow and goes beyond 100% in some cases, as seen in Figure 2(b). For example with latencies of 256 ms at 8 Mbps, Cascaded TCP continues to be twice as efficient as compared to long-haul TCP. The same is observed for bandwidths as high as 32 Mbps and latencies 64 ms, which is in fact the same bandwidth-delay product as the former example. We see similar trends with varying losses too.

We observe smooth trends in our results except for select cases in Figure 2(b) of 32 Mbps capacity and 128 ms latency. This apparent anomaly is due to the observed value being just over the lower bin limit and hence an artifact of binning.

We note that throughput drops at high bandwidth-delay products and we see diminishing returns with one relay. We anticipate that if more relays are added, while assuming that latency is equally split between them, we would continue to see increasing utilizations.

3) Multiple relays continue to improve throughput: Figure 3 shows that using two relays results in further improvement in throughput and therefore utilization. With two relays, Cascaded TCP achieves approximately 90% utilization when the end-to-end latency is 64 ms and loss is 0.1%. This is reasonably near the theoretical limit of about 94% when compared to a connection with maximum utilization — the limit of 94% is experienced due to protocol overheads. We see the same trends for all cases of losses. Here again, we see that median bandwidth utilizations for Cascaded TCP are always significantly better than that of long-haul TCP.

4) Cascaded TCP performs well with high losses: With Cascaded TCP, we achieve better throughput even when losses are high because the latency for the split TCP connections is shorter, allowing the congestion-control algorithm to react more quickly. This delivers better throughput and therefore better bandwidth utilization. With zero losses, Cascaded TCP enables throughput up to twice as much as long-haul TCP connections; this is observed even when latencies are as high as 256 ms. Here we do not show bandwidth utilization results for varying losses due to lack of space.

Note that for low loss, the improvement in throughput for two relays over one relay is not of the same magnitude as it

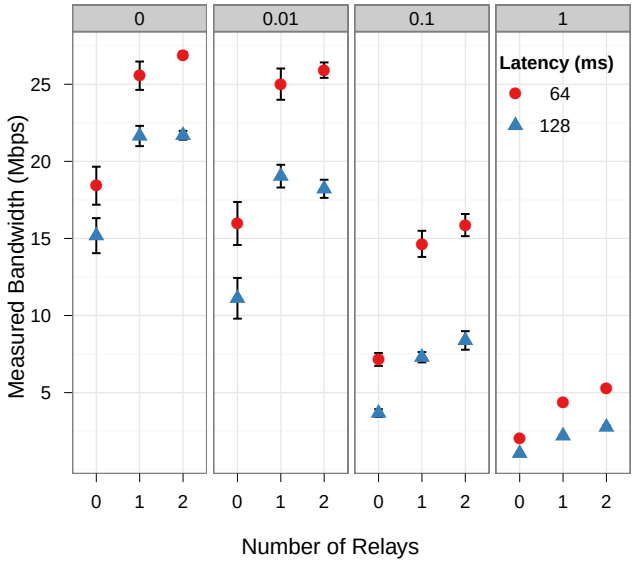


Fig. 3. Results for link capacity of 32 Mbps, latencies of 64 ms and 128 ms and losses of 0%, 0.01%, 0.1% and 1% are presented to compare long-haul and cascaded TCP with one and two relays. Zero relays imply long-haul TCP.

is for one relay over no relay (or long-haul TCP), as shown in Figure 3. However, as loss increases, we see that the magnitude of improvement is similar for two relays as compared to one and one relay as compared to none. At high losses, the congestion-control protocol does not allow the window sizes to grow because of losses and therefore we are able to see the benefits of having relays. The more relays we add, the more we alleviate the impact of losses. However at low losses, throughputs increase up to link capacity and are gated and thus the benefits of increasing the number of relays is less evident. In Figure 3, for 0.01% loss, we see that the average measured throughput for 128 ms latency configuration is less for two relays as compared to one relay. This anomaly is because of temporary self-congestion induced by the sender. Apart from this anomaly, we did not observe this behavior for other configurations.

5) Cascaded TCP measurements correlate with Mathis' model: Figure 4 shows measured bandwidths with respect to Mathis' approximation of the upper limit on bandwidth. We see that our empirical results for long-haul TCP have a strong correlation with Mathis' model, until we hit the link capacity — where throughput is capped. In other words, empirical observations show that beyond certain percentages of available bandwidth, the link capacity starts becoming bottleneck. As mentioned earlier, when throughputs are gated by link capacity, the measurements are reasonably near the theoretical limit of about 94%.

Cascaded TCP has a steep slope, indicating that Cascaded TCP results in better throughput than what Mathis' model predicts, based on the end-to-end latency. The steep slope is observed because Cascaded TCP effectively doubles the achieved bandwidth by allowing the congestion-control algorithm to react faster; this assumes that the link capacity supports higher bandwidths.

We observe that throughputs experienced by a long-haul TCP connection for a particular configuration are about half

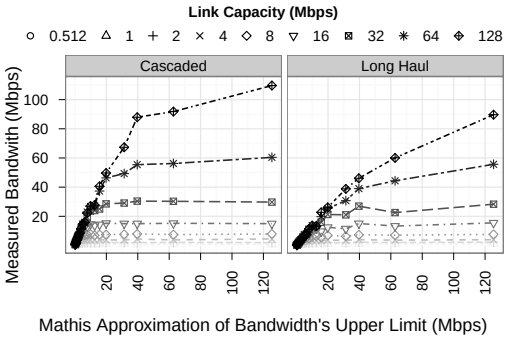


Fig. 4. Measured bandwidths are analyzed with respect to Mathis' approximation of upper limit on bandwidth when using TCP.

of that observed by Cascaded TCP (with one relay). This effect is profound at larger latencies (e.g., 128 ms and 256 ms). We discuss the reasons for this behavior in Section VI-B.

B. Case Study using PlanetLab

We conducted a case study on real networks using the PlanetLab testbed. We evaluated the use of a single relay on different network paths. These paths included inter-continental links. Our findings across these network paths were similar.

Note that with PlanetLab, as we operate in the live network, the cross traffic inhibits optimal bandwidth utilization. Also, there may be unknown circumstances (e.g., asymmetric link capacities) and events that may or may not influence the behavior of the traffic flowing across.

TABLE II
FINDINGS FROM A SELECT CASE STUDY

Metric	Value
Mean latency, client to server	$448.07ms \pm 1.6ms$
Mean latency, client to relay	$194.74ms \pm 0.58ms$
Mean latency, relay to server	$295.06ms \pm 0.34ms$
Packet loss, all paths	0%
Long-haul bandwidth	$\mu = 5.85Mbps,\ CI : (5.25, 6.39)$
Pipelined bandwidth	$\mu = 6.44Mbps,\ CI : (6.00, 6.98)$
p-value, $\mathcal{H}_0 : \mu_p = \mu_{lh}$	0.011^*

Consider the path: from planetlab1.iitkgp.ac.in, via planetlab-1.imperial.ac.uk, to planetlab1.sfc.wide.ad.jp. The performance measurements are summarized in Table II. We see that the difference in throughput achieved by Cascaded TCP as compared to long-haul TCP is statistically significant. Here the difference is limited to about 10%, which can be explained by the use of default window sizes for TCP connections from the hosts. We were unable to reconfigure the hosts to use larger window sizes due to limited administrative access. These default window sizes gated the sender window sizes from growing to accommodate the larger bandwidth-delay product. This resulted in lesser gains as compared to experiments in an ideal environment.

VI. Discussion

A. When does Cascaded TCP become viable?

As observed in Figures 2(a) and 2(b), pipelined Cascaded TCP is viable for almost all configurations except for small bandwidth-delay products. In Figure 2(a), we see that a single relay continues to be efficient for bandwidths as high as

64 Mbps (at 64 ms) and 128 Mbps (at 32 ms). Beyond that, to remain efficient (i.e., obtain greater than 80% utilization), a second relay is needed, as illustrated in Figure 3.

As discussed in Section III-E using (5), (6) and (7), we can evaluate condition (3), which allows us to determine if using Cascaded TCP would be feasible. We compute estimates for throughput using the Cascaded TCP model and compare them with measured throughput. Results for 128 Mbps link capacity are presented in Figure 5.

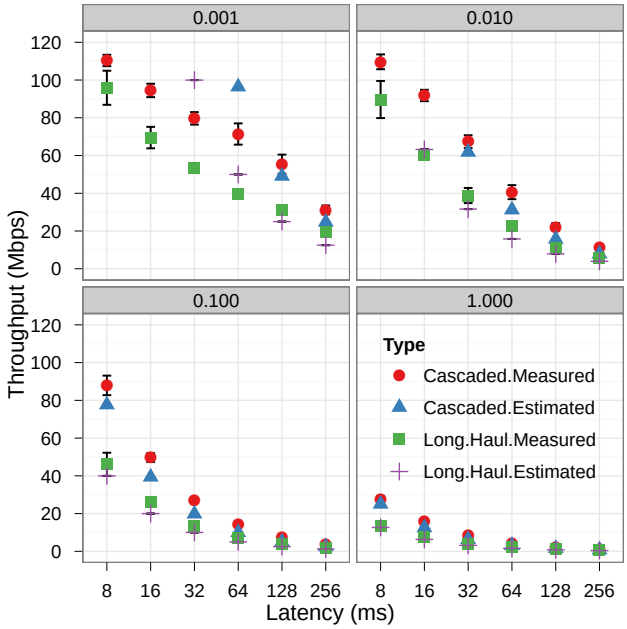


Fig. 5. Estimated and measured throughput results are presented for link capacity of 128 $Mbps$ with losses of $0.001\%, 0.01\%, 0.1\%$ and 1%. In the model we use loss of 0.001% to approximate 0% loss. Note that bandwidth-delay products are proportional to latencies which are shown in this figure.

We see that the analytical model provides acceptable approximations for achievable bandwidth. The errors can be partly explained by the simplifying assumptions we use to approximate overheads. Note that Mathis' model is also an approximation upon which we base our model. If Cascaded TCP were based on a more precise model (e.g., [17]) it would yield better estimates. Nevertheless the analytical model allows us to make an informed decision whether to use Cascaded TCP or not. As with Mathis' model, the predictions beyond link capacity are not valid and for negligible losses (i.e., 0.001%) the TCP model overestimates throughput (e.g., throughput estimates for 32 ms latency and 0.001% loss in Figure 5).

The relative differences computed from empirical results (also shown in Figure 2(b)) for the same configuration highlight that we achieve approximately 100 percent improvement, which is what the model predicts.

In contrast to large bandwidth-delay product scenarios, we note that the model predicts minimal improvement in throughputs for low bandwidth-delay products.

In summary, we observe that Cascaded TCP is beneficial when the bandwidth-delay product is greater than 32 KB, which incidentally is less than the the typical default limit for window sizes — FreeBSD and other operating systems typically have 64 KB as the default TCP window size.

B. How many relays do we need?

We can modify Mathis' throughput approximation to accommodate relays. If we assume homogeneity, the throughput would be as shown in (8), suggesting that the maximum throughput achieved would be that of the bottleneck. We may represent the latency for the bottleneck link as $RTT/(N+1)$ and loss as $p/(N+1)$, where N is the number of relays and for the sake of simplicity, loss is assumed to remain the uniform across links. Therefore, we have:

$$throughput = \left(\frac{MSS}{\frac{RTT}{N+1}}\right)\left(\frac{1}{\sqrt{\frac{p}{N+1}}}\right) = \left(R^{3/2}\right)K, \quad (8)$$

where $R = N+1$, and $K = \frac{MSS}{RTT\sqrt{p}}$. Subsequently,

$$\frac{\partial B}{\partial R} = \frac{3}{2}R^{1/2}K. \quad (9)$$

Equation (9) is a monotonically increasing function. This implies that if we continue to increase the number of relays that we should expect bandwidth utilization to increase until the capacity limits are reached. In Figure 4, we see that if we introduce a relay, we effectively double the throughput. This is until the link capacity is reached. While this approximation may be true theoretically, it is not so practically. From a cursory case study of available locations to setup relays between Virginia and New Mexico we concluded that we would be able to setup at most six to eight relays.

C. Where should the relays be located?

For simplicity we assume uniform spacing for the relays — for example one relay is setup half way between the sender and the receiver, similarly two relays are located such that the latency for each layer-4 hop is about one third the end-to-end latency between the sender and the receiver. This may not be a practical assumption. Our model and empirical results show that maximum improvement in performance is experienced when the relays are equally spaced. If they were to be placed close or farther to the source the benefits would be reduced. In this case the bottleneck bandwidth will be dictated by the segment with the longer RTT and the benefits would decrease in proportion to the ratio of the longer segment's RTT and the long-haul connection's RTT.

We also assume that the relays are located along the same path the long-haul TCP traffic traverses. It may not be possible to always locate a relay along the same path the long-haul traffic traverses. Also, an alternate path may have different loss, latency and capacity configurations therefore incur additional overheads.

D. Does Cascaded TCP maintain end-to-end semantics?

The end-to-end semantics of TCP are broken when using Cascaded TCP because the connection is split into a cascade of independent TCP connections, which are put together by the relays to form a logical connection. We note that end-to-end

semantics are also broken by middleboxes such as firewalls and NATs. The concerns we face in maintaining end-to-end semantics with Cascaded TCP relays are no more than what we already experience with middleboxes. As long as the risks of using such relays/middleboxes are understood, the benefits of increased throughput outweighs the concerns. Nevertheless this may not apply to all situations.

E. Does Cascaded TCP maintain TCP friendliness?

The expectations from congestion control algorithms are that they maintain high bandwidth utilization, RTT fairness and end-to-end semantics. They are also expected to be TCP friendly. As Cascaded TCP continues to use TCP New Reno, it maintains both TCP friendliness and RTT fairness while improving network throughput.

BIC, CUBIC and Compound TCP do maintain end-to-end semantics and improve upon bandwidth utilization (when compared with TCP New Reno), however, they do not maintain TCP friendliness and RTT fairness for large latencies [9].

F. Should the use of Cascaded TCP be hidden from the user and how can it be deployed?

Whether the use of Cascaded TCP relays is apparent to applications or not depends on how the transport layer implements the solution. What is important to note is that there is a need to make a decision based on context in which communication occurs — for example based on latency, loss, link capacity experienced by the connection and availability of relays. This decision process may be transparent and automated by a daemon implementing the analytical model as described in Section III, subsequently aiding the network stack. or it may be controlled by the end user as was suggested by Border et al. [4] by implementing policies.

As explained in Section IV we used an expedient method to establish a proof of concept. Ideally, to support the transport layer a framework would be required to identify potential relays, setup transport connections between the relays and manage communication.

VII. Future Work

We plan to investigate how cascaded TCP, which we've seen to be a viable step towards improving throughput, performs in connection with other congestion control algorithms (such as [6], [7]) — to study the benefits of using them together and as an alternate solution. We also intend to study the benefits of Cascaded TCP for parallel streams.

If the sender does not generate traffic at a suitable rate the relays may experience buffer drought. We try to accommodate for this aspect in our model (in (6)) by considering the number of windows that the relay waits for before it may start relaying. Alternatively, if the sender generates traffic at a higher rate than what the relay could accommodate, we face a buffer overflow problem. We are in the process of formulating an optimization problem that allows us to choose the location of relays and buffer sizes when considering the latency and loss constraints while maximizing throughput.

VIII. Summary

In this paper, we have shown that we can improve bandwidth utilization by reducing the impact of end-to-end latency on typical congestion-control protocols. We do so by introducing layer-4 forwarding relays, which allows us to split a TCP connection into two or more TCP connections that are cascaded together to form one logical end-to-end connection. As each segment's congestion-control algorithm operates independently and reacts to feedback from the receiver much faster than that of the long-haul TCP connection, Cascaded TCP enables greater overall throughput and thus better bandwidth utilization. We present an analytical model that allows us to make an informed decision as to when the use of Cascaded TCP would be viable. We present and evaluate the results of the analytical model and our empirical tests. We conclude that introducing relays results in a significant increase in throughput for many practical scenarios.

Acknowledgements

This research was funded in part by Virginia Tech and Juniper Networks.

References

[1] S. Floyd *et al.*, "The NewReno Modification to TCP's Fast Recovery Algorithm," RFC 3782 (Proposed Standard), 2004.
[2] W. Feng, "Long-Haul TCP vs. Cascaded TCP," Computer Science, Virginia Tech, Tech. Rep. TR-06-04, 2006. [Online]. Available: http://eprints.cs.vt.edu/archive/00000737/
[3] S. Chesire, "It's the Latency, Stupid." [Online]. Available: http://rescomp.stanford.edu/~cheshire/rants/Latency.html
[4] J. Border *et al.*, "Performance Enhancing Proxies Intended to Mitigate Link-Related Degradations," RFC 3135 (Informational), 2001.
[5] E. Nygren *et al.*, "The Akamai Network: A Platform for High-Performance Internet Applications," *SIGOPS Operating Systems Review*, vol. 44, no. 3, 2010.
[6] L. Xu *et al.*, "Binary Increase Congestion Control (BIC) for Fast Long-Distance Networks," in *IEEE INFOCOM*, 2004.
[7] S. Ha *et al.*, "CUBIC: A New TCP-Friendly High-Speed TCP Variant," *ACM SIGOPS Operating Systems Review*, vol. 42, no. 5, 2008.
[8] K. Tan *et al.*, "A Compound TCP Approach for High-Speed and Long Distance Networks," Microsoft Research, Tech. Rep. TR-2005-86, 2005.
[9] S. Bhatti *et al.*, "Transport Protocol Throughput Fairness," *Journal of Networks*, vol. 4, no. 9, 2009.
[10] Y. Zhang *et al.*, "An Implementation and Experimental Study of the Explicit Control Protocol (XCP)," in *IEEE INFOCOM*, 2005.
[11] K. Ramakrishnan *et al.*, "The Addition of Explicit Congestion Notification (ECN) to IP," RFC 3168 (Proposed Standard), 2001.
[12] H. Sivakumar *et al.*, "PSockets: The Case for Application-Level Network Striping for Data Intensive Applications Using High Speed Wide Area Networks," in *IEEE/ACM Supercomputing*, 2000.
[13] M. Zhang *et al.*, "A Transport Layer Approach for Improving End-to-End Performance and Robustness Using Redundant Paths," in *USENIX ATC*, 2004.
[14] X. Fang *et al.*, "A Hybrid Network Architecture for File Transfers," *IEEE TPDS*, vol. 20, no. 12, 2009.
[15] H. Y. Pucha *et al.*, "Slot: Shortened Loop Internet Transport using Overlay Networks," Purdue University, Tech. Rep. TR-ECE-5-12, 2005.
[16] M. Mathis *et al.*, "The Macroscopic Behavior of the TCP Congestion Avoidance Algorithm," *ACM SIGCOMM CCR*, vol. 27, no. 3, 1997.
[17] J. Padhye *et al.*, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," in *ACM SIGCOMM*, 1998.
[18] Tirumala, Ajay and Gates, Mark and Qin, Feng and Dugan, Jon, "Iperf - The TCP/UDP Bandwidth Measurement Tool." [Online]. Available: http://dast.nlanr.net/Projects/Iperf
[19] G. Giacobbi, "Netcat - The TCP/IP Swiss Army." [Online]. Available: http://nc110.sourceforge.net/
[20] L. Cottrell, "Ping End-to-End Reporting." [Online]. Available: http://www-iepm.slac.stanford.edu/pinger/