Rethinking the Timescales at Which Congestion-control Operates

Vishnu Konda and **Jasleen Kaur**
University of North Carolina at Chapel Hill
{vishnu, jasleen}@cs.unc.edu

Abstract—**The efficiency of TCP congestion-control in achieving high throughput is quite poor in high-speed, lossy, and dynamic-bandwidth environments. The main culprit is the *slow bandwidth-search process* used by TCP, which may take up to several thousands of round-trip times (RTTs) in searching for and acquiring the end-to-end spare bandwidth. While several alternate protocols have been proposed to speed up the search process, these still take hundreds of RTTs for doing so.**

In this paper, we argue that the sluggishness of existing protocols stems from two limiting design decisions that help a transfer remain non-intrusive to competing transfers. We argue that these legacy design decisions can be done away with if we *limit the impact of probing for spare bandwidth*. We use this idea to design a new approach for congestion-control that allows TCP connections to boldly search for, and adapt to, the available bandwidth within a single RTT. Our approach relies on carefully orchestrated packet sending times and estimates the available bandwidth based on the delays experienced by these. We instantiate our new protocol, referred to as RAPID, using mechanisms that promote efficiency as well as queue-friendliness. Our experimental evaluations indicate that RAPID: (i) converges to an updated value of bandwidth within 1-2 RTTs; (ii) helps maintain fairly small queues even in high-speed networks; and (iii) has negligible impact on regular TCP traffic. The benefits of our approach are especially significant on lossy links and those with rapidly-changing bandwidth.[1]

I. Introduction

"Congestion Control" can be easily listed among the top-10 networking problems of the past two decades. And indeed, why not? A congestion-control protocol has no simple task—it has to discover the end-to-end spare bandwidth available to a transfer in a quick, adaptive, and non-intrusive manner. Simultaneously achieving these properties turns out to be a significant challenge, especially for an end-to-end protocol that receives no explicit feedback from routers/switches. Indeed, the dominant end-to-end transport protocol, TCP NewReno, has been shown to be abysmally slow in discovering the spare bandwidth, especially in high-speed networks and on paths that experience dynamic bandwidth and non-congestion losses.

Several alternate protocols have been proposed to address this limitation. However, as discussed in Section II-A, most of these protocols struggle to remain non-intrusive to other network traffic while achieving speed—consequently, we believe that these designs are still quite sluggish in probing for spare bandwidth. In particular, even the so-called "high-speed" protocols may still take 100s-to-1000s of round-trip times (RTTs) to converge to a stable sending rate.

In this paper, inspired by recent advances in the field of bandwidth estimation, we propose the idea that the sluggishness of transport protocols can be done away with without overloading the network, if we *limit the impact of probing for spare bandwidth*. We use this idea to design a novel approach, referred to as *RAPID Congestion Control (RAPID)*, that exhibits two significantly desirable characteristics. Most notably, it reduces the time it takes a transport protocol to acquire freshly-available bandwidth by more than an order of magnitude. Equally significantly, by relying on a delay-based congestion-detection strategy, the protocol ensures that it is friendly to regular TCP transfers, and that packet queues at bottleneck links are small and transient.

In the rest of this paper, we discuss the sluggishness of existing protocols and present our key insight in Section II. We describe the RAPID protocol mechanisms in Section III and present preliminary performance evaluation results in Section IV. We conclude in Section V.

II. Congestion-control Timescales

A. The Problem: Slow Feedback Loop

The basic issue considered in this work is that of the sluggishness of transport protocols in discovering the available bandwidth (also referred to as *avail-bw* or AB). To understand the issue, consider how an end-to-end transport protocol probes for the avail-bw in the absence of explicit feedback from the network. In steady-state, the protocol continuously operates a 2-phase AB-search cycle: in the *search-phase*, it successively "tries out" larger data sending rates. For each rate probed at, it examines performance feedback such as packet loss and high end-to-end delays that arrives after an RTT-worth of delay—this helps it estimate when the most recent sending rate was higher than the avail-bw.[2] When such a rate is reached, the *reduction-phase* of the protocol reduces the sending rate to a lower value and switches back to the search-phase. The speed with which each loop of this search-cycle is executed fundamentally determines how quickly a protocol can acquire and adapt to changes in the avail-bw.

[1]This research was supported in part by NSF CAREER award CNS-0347814 and NSF RI grant EIA-0303590. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

[2]Different protocols differ in how the successive probing rates relate to the current rate, as well as in the performance measures they use as feedback. Table I summarizes these differences for some prominent protocols.

Protocol	Search-phase (per-RTT increase in probe-rate)	Feedback-metric	Experimentally-observed time to acquire AB = 1 Gbps
NewReno	Additive Increase	Packet Loss	$\sim$ 7433 RTT
HighSpeed	Multiplicative Increase	Packet Loss	$\sim$ 666 RTT
FAST	Multiplicative Increase	Packet Delays (and Loss)	$\sim$ 70 RTT
RAPID	Exponential Increase	Delays (and Loss)	$\sim$ 4 RTT

TABLE I
TIME TAKEN TO ACQUIRE AN AB OF 1 GBPS BY DIFFERENT PROTOCOLS (SEE SECTION IV)

Note that the smallest timescale at which the above cycle could be executed is a round-trip time (RTT)—performance feedback can arrive no earlier than this time. Unfortunately, most existing protocols execute this cycle at timescales much larger than this. This is because, primarily driven by the goal of not overloading the network, all previous protocols adopt two limiting design features:

1) *Only a single (larger) sending rate is probed for over a round-trip time.*
This is true for all previous protocols, including recent ones, such as HighSpeed TCP, FAST, Scalable, CUBIC, and PCP [1], [2], [3], [4], [5], [6], [7]. This legacy design decision is perhaps motivated by the fact that unless the single rate is deemed acceptable (not too high), other rates should not be probed for.
2) *The new rate probed for is not significantly larger than the previous sending rate.*
This feature is adopted primarily to prevent a single transfer from overloading the network, in case the previous rate was quite close to the avail-bw. Existing protocols differ in how the new probing rate relates to the previous rate—for most protocols, the ratio of (probeRate - previousRate)/previousRate is bounded by a small fraction of 1.[3]

As a result of these two design limitations, most protocols—even the recent ones designed for high-speed networks [2], [3], [4], [5], [6], [7]—take a fairly long time for converging to the avail-bw. For instance, Table I lists the experimentally-observed times taken by a single transfer for acquiring an additional spare capacity of 1 Gbps that suddenly becomes available.[4] We find that even high-speed protocols can take hundreds of RTTs for converging to the avail-bw.

B. Key Insight: Limit Probing Volume

We believe that both of the above design limitations can be done away with if one *limits the volume (and impact) of probing for a larger sending rate*. Specifically, we advocate that a transport protocol should: (i) use a rate-based packet transmission mechanism, and (ii) rely on packet delays for estimating avail-bw within an RTT. We believe that such a protocol can, in fact, boldly probe for an exponentially-wide range of candidate sending rates within a single RTT—any associated overloading impact can be avoided by using the following guidelines:

1) *Achieve a rapid AB search:* Probe for an *exponentially-wide* range of candidate sending rates within a single RTT. However, send extremely small probes (of one packet each) at each candidate rate in order to limit the transient overloading impact of the large rates.
2) *Avoid persistently overloading the network path:* Ensure that the *average* rate of packet transmission does not exceed the most-recently discovered estimate of avail-bw. This implies that some of the rates in the above-suggested exponential range will be smaller than this estimate, and some will be larger.

We use these ideas to design a new protocol, referred to as RAPID Congestion Control (RAPID), and show that it can adapt to fairly large changes in AB within 1-4 RTTs. Furthermore, RAPID can avoid persistent or large router queues due to its efforts in avoiding network overload. The benefits of the protocol are especially significant in dynamic bandwidth environments and high-speed networks.

It is worth mentioning that the PCP protocol proposed in [1] also relies on a rate-based transmission and adopts a guideline similar to the second one advocated above. However, like all other existing protocols, this protocol also probes for only a single probing rate within an RTT. The prime advantage of this protocol is that it is significantly less intrusive than most window-based protocols; our initial investigations, however, suggest that even PCP is more aggressive and intrusive to on-going transfers than RAPID. Nevertheless, PCP comes closest to RAPID in spirit—we are currently conducting experimental evaluations to compare the two protocols.

We present the basic mechanisms used in RAPID in Section III and some preliminary evaluations in Section IV.

III. RAPID CONGESTION-CONTROL

Unlike many congestion-control protocols, RAPID employs a rate-based transmission policy and relies on packet delays for estimating avail-bw—its mechanisms, therefore, are designed quite differently from those of most protocols. While the RAPID design is motivated by the primary goal of shrinking the timescales at which congestion-control operates, several equally-important goals are given due consideration in the design process [8]. Most significantly, a RAPID network strives to (i) maintain a *low buffer occupancy* at congested router links, and (ii) remain *friendly to regular TCP transfers*.

[3]The exception is CUBIC, in which a binary search method is used after the AB is discovered for the first time—here, the ratio probeRate/previousRate depends on the past probing history.

[4]These experiments were run on the ns-2 simulator, and a packet size of 1040 B was used—see Section IV.

Below, we describe the basic mechanisms used for achieving each of these.

A. Acquiring AB Within a Few RTTs

As long as there is data to send, a RAPID sender continuously transmits data in logical groups of N packets each, referred to as a *multi-rate probe stream (p-stream)*. The i-th packet in a p-stream is sent at a rate of r_{i-1}; this implies that the transmission times of packets i and $i-1$ differ by $\frac{P}{r_{i-1}}$, where P is the size of packet i. The sender controls the average sending rate of a p-stream, referred to as r_{avg}, which is given by: $\frac{N-1}{r_{avg}} = \frac{1}{r_1} + \frac{1}{r_2} + \ldots + \frac{1}{r_{N-1}}$. Further, for all $i > 1$, $r_i > r_{i-1}$.[5]

a) AB-estimation logic: We use the one-way delays experienced by packets in a p-stream for estimating avail-bw in the same manner as the PathChirp bandwidth estimation tool [9]. When the RAPID receiver receives all packets of a p-stream, it computes the AB by looking for increasing trends in the one-way delays experienced by packets. Intuitively, if i is the first packet in a p-stream such that $r_{i-1} \geq AB$, then each of the packets $i, \ldots, N$ will queue up behind its previous packet at the bottleneck link—due to this "self-congestion", each of these packets will experience a larger one-way delay than its predecessor. Thus, the smallest rate at which the receiver observes an increasing trend in one-way delays can be used as an estimate of the current avail-bw[6]—we refer the reader to [9] for details and the precise formulation.

The receiver communicates the avail-bw estimate, henceforth referred to as $ABest$, to the sender.

b) Transmitting in a non-overloading manner: When the sender receives an $ABest$, it updates the r_{avg} of the next p-stream as: $r_{avg} = ABest$. Thus, the transfer acquires a sending rate equal to the avail-bw within an RTT. It then selects an appropriate set of rates, $r_1, \ldots, r_{N-1}$, such that the average of these is equal to r_{avg}. Thus, even though a p-stream probes for new rates of up to r_{N-1}, which may be much larger than $ABest$, the *average* sending rate of a p-stream does not exceed this value. This helps ensure that the average load on the bottleneck link does not exceed its capacity—this is critical for maintaining small and transient queues at the bottleneck links.

c) Speeding up the search process: Each p-stream probes for the range of sending rates given by: $[r_1, \ldots, r_{N-1}]$. The larger is the ratio of $\frac{r_{N-1}}{r_1}$, the faster is the AB-search process.

Note that for a given r_{avg} and N, there are infinite choices for the set: $r_1, \ldots, r_{N-1}$. So for instance, while these rates could be additively-related as in: $r_i = r_1 + (i-1)\delta$, a faster search will be obtained by using a multiplicative-relation as in: $r_i = m^{i-1} * r_1$. The current version of RAPID adopts this latter relation using a multiplicative factor of $m = 1.07$, and $N = 22$. This yields: $r_1 \approx 0.6 * r_{avg}$ and $r_{N-1} \approx 2.5 * r_{avg}$.

With the above choice of rates, RAPID can probe for avail-bw spanning several orders of magnitude within a few RTTs.

[5]The gap between the first packet of a p-stream and the last packet of the previous p-stream is set to r_{avg}. This can also be stated as: $r_0 = r_{avg}$.

[6]If no increasing trend is detected, r_{N-1} is taken as the AB estimate.

d) Achieving a Quick-yet-Slow Start: RAPID faces a similar dilemma as all congestion-control protocols—how to obtain the initial $ABest$ (or the initial r_{avg}) for a new transfer? We address this issue with the same approach as most other protocols—fortunately, our ability to probe for multiple rates within an RTT makes our slow-start much faster than other protocols. Specifically, when a new transfer begins, we initialize r_{avg} to a small value (currently, around 850Kbps) and set the multiplicative factor as: $m = 2$. Since even this relatively-small rate may overload an already-congested path, we limit the adverse impact by initializing $N = 4$. If the $ABest$ returned is no smaller than r_3, we increase N to 8 and send another p-stream with the new r_{avg}. We repeat this process—and increase N multiplicatively till a maximum value of 64—till we get an $ABest$ within the range of rates probed for.[7] Following this, we switch to the steady-state RAPID mode, in which $m = 1.07$ and $N = 22$.

B. Remaining TCP-Friendly

RAPID is quite non-intrusive to regular TCP NewReno transfers. The prime reason for this is that the RAPID congestion-control relies on queuing delay as feedback, whereas TCP reduces its sending rate only on witnessing packet losses. When a router carrying both TCP and RAPID transfers gets congested, the RAPID transfers would respond to the congestion and reduce their sending rates much before the TCP transfers would. The downside is that in the presence of long-lived TCP transfers, RAPID transfers would obtain much lower throughput than the former. However, this problem plagues any network that runs fundamentally different congestion-control protocols—an easy solution is to provision routers with separate queues for traffic from different protocols.

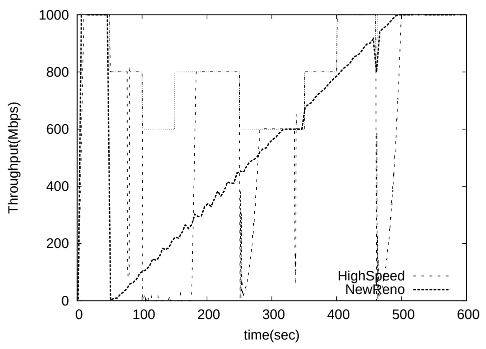
We next experimentally evaluate how well the above mechanisms achieve the stated goals for RAPID.

IV. Experimental Evaluation

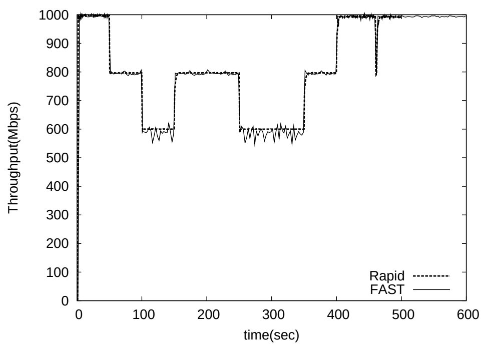
We use the ns-2 simulator for evaluating the performance of RAPID and other prominent congestion-control protocols. For all of our simulations, we rely on a simple dumbbell topology in which multiple sources and aggregated at a single bottleneck link—this bottleneck link is the only link shared by co-existing transfers. Unless noted otherwise, the bottleneck link is provisioned with a delay-bandwidth product (DBP) worth of buffers, where the delay is the average end-to-end propagation delay for transfers (set to 60 ms). We set the maximum size of each link-layer packet to 1040 B.

In what follows, we summarize our experiments and observations.

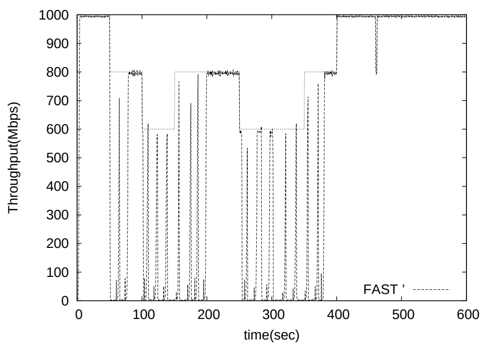
[7]Note that this process is no more aggressive than the slow-start adopted by most protocols, which multiplicatively increases the number of packets sent over an RTT in exactly the same manner. In fact, by setting $r_{avg} = ABest$, we ensure that any overload is merely transient. Also note that 64 is a commonly-adopted setting for the slow-start threshold.



(a) NewReno, HighSpeed



(b) FAST, RAPID



(c) FAST'

Fig. 1. Speed of Acquiring AB (Dynamic Bandwidth)

A. Speed of Acquiring Spare Bandwidth

TCP is known to be quite inefficient in utilizing spare bandwidth in two kinds of environments: (i) in high-speed networks, especially those that witness sudden and large changes in avail-bw, and (ii) in networks that experience non-congestion losses induced by bit-errors (such losses can occur on most transmission mediums, though with significantly different probabilities). We evaluate NewReno, HighSpeed, FAST, and RAPID by simulating examples of these two cases. For all of the experiments in this section, we simulate a 1Gbps bottleneck link in our dumbbell topology.

e) High-speed Networks with Dynamic Bandwidth: In the first set of experiments, we simulate a network for 600 seconds, and introduce 4 constant-bit-rate (cbr) traffic streams on the bottleneck link according to the following schedule: *cbr-1* exists from 50-400 seconds, *cbr-2* exists from 100-150 seconds, *cbr-3* exists from 250-350 seconds, and *cbr-4* exists for a small duration from 460-462 seconds. Each cbr stream has a bit-rate of 200 Mbps. The spare bandwidth left on the network is plotted in Figs 1(a)-(c) using a faint dotted line.

We use this setup to run a series of experiments in which we introduce a single long-lived transfer at time 1 second, and respectively, run it over NewReno, HighSpeed, FAST, and RAPID. The throughput obtained by the transfer in each experiment is also plotted in Figs 1(a) and 1(b). We find that:

1) NewReno and HighSpeed, which are both loss-based protocols, experience packet losses when the avail-bw reduces suddenly. This is because a loss-based protocol induces persistent queuing in the bottleneck buffers (see queue-size distribution in Fig 2)—any sudden decrease in AB overflows the buffers causing multiple packet losses. When this happens, NewReno takes an abysmally long time (more than 7000 RTTs) to re-acquire the spare bandwidth. HighSpeed is faster, but still takes up to 700 RTTs to recover its throughput.
2) Delay-based FAST performs much better since it does not overflow router buffers and does not witness multiple losses. In fact, FAST maintains a smaller but significant number of packets in the bottleneck queue and is able to utilize sudden increase in AB as well.

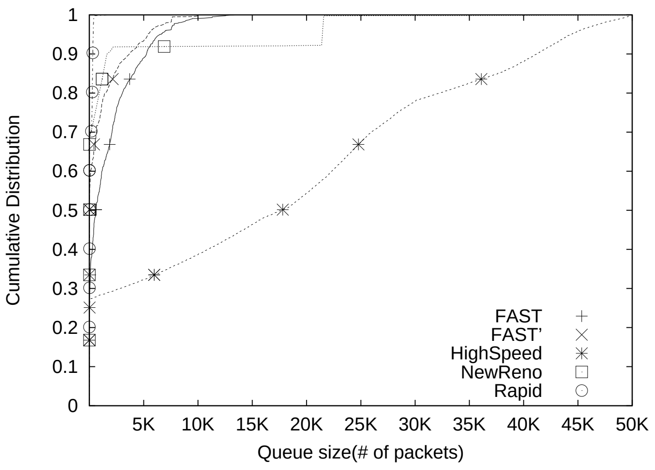
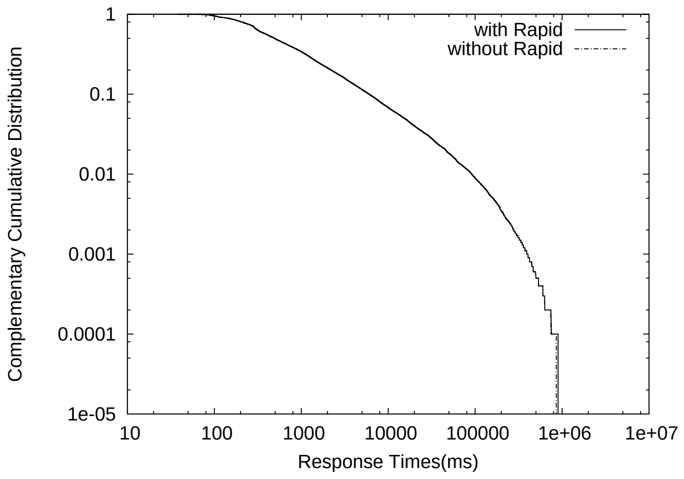


Fig. 2. Queue Size Distribution (Dynamic AB)

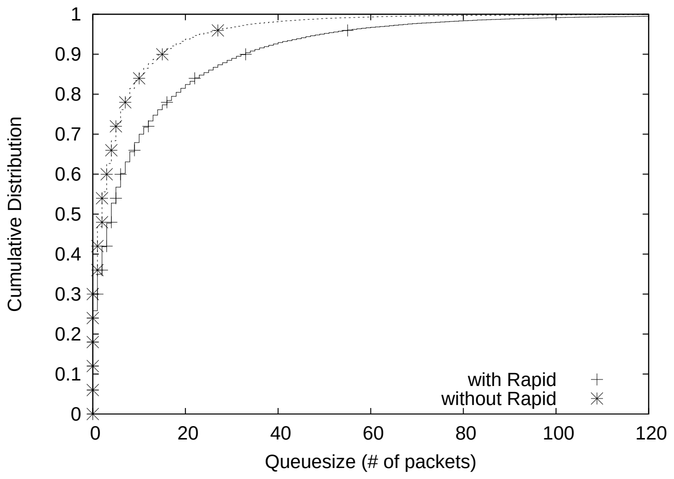
Fig 2 shows that queues can grow to more than 10,000 packets in the FAST experiment. In order to see how FAST would perform in networks with limited buffers, we provision the bottleneck link with exactly 10,000 buffers (which is around 15% of the DBP) and re-run the FAST experiment (plotted as *FAST'* in Fig 1(c)). We expect to see some packet loss in this case—indeed, Fig 1(c) shows that FAST' does witness multiple packet losses initially; surprisingly, however, it is simply unable to re-stabilize to the spare bandwidth for several hundred seconds.

f) Non-congestion Error-based Losses: Error-based losses can occur with probabilities of up to 10^{-6} even on reliable optic fiber links. In high-speed networks, this is not ignorable since a 1Gbps transfer would witness losses at timescales of tens-to-hundreds of seconds. In such a setting, the ability of a congestion-control protocol to re-acquire AB quickly after a loss is critical.

In order to study this ability, we simulate a periodic error loss process on the 1Gbps bottleneck link—this process periodically drops 1 in 10^6 packets. We simulate a single long-lived transfer on this topology for 600 seconds, using HighSpeed, FAST, and RAPID, respectively, as the underlying protocols. Figure 3 plots the resultant throughput (observed in the period 30-70 seconds). We find that each transfer



(a) Response Times of Tmix Transfers



(b) Router Queue Sizes

Fig. 4. Impact on Tmix-generated TCP Traffic

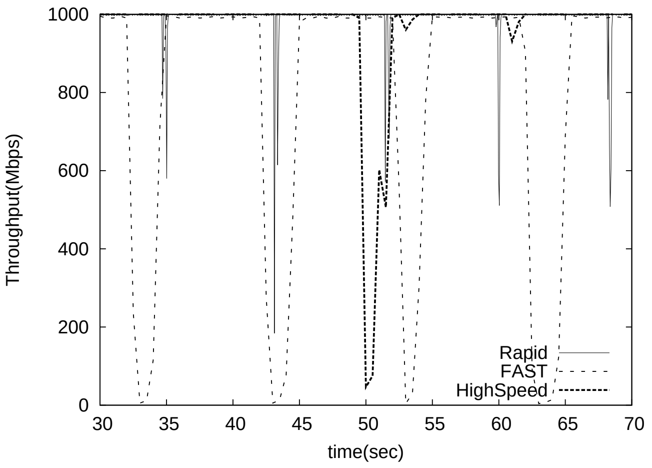


Fig. 3. Speed of Acquiring AB (Error Losses)

experiences around 4-5 losses in this duration. FAST takes a significant amount of time (around 70 RTTs) to re-acquire the gigabit bandwidth after a loss. HighSpeed fares much better (except in one case of loss) due to its tendency to build-up very large router queues that help it maintain high throughput while it recovers from the packet losses—on the flip side, router queues grow undesirably large. In contrast, RAPID is able to quickly recover from each loss within a few RTTs, while also maintaining negligible queues.

B. Impact on TCP Traffic

Finally, we evaluate the impact of high-throughput RAPID transfers on a realistic mix of regular TCP transfers that co-exist on a bottleneck link. For this, we use the publicly-available Tmix traffic generator code for ns-2, which generates an empirically-derived TCP traffic mix [10]. We generate traffic at an average offered load of 90Mbps for 40 minutes and drive it through a bottleneck link of 120 Mbps. We log the time it takes to complete each TCP transfer—referred to as the *response time* of the transfer. We collect logs after 20 minutes of simulation time (to allow the traffic-generator to reach a steady-state of connection arrivals and departures).

We then re-run the Tmix experiment with a long-lived RAPID transfer sharing the network. In Fig 4(a), we plot the complementary cumulative distribution of response times of the TCP transfers in both experiments. We find that the presence of an RAPID transfer has a visually-indistinguishable impact on the response times of both short and long TCP transfers. We also find that the RAPID transfer is able to well-utilize the spare bandwidth—it achieves nearly 100% link utilization. Fig 4(b), which plots the distribution of queue sizes in the two experiments, shows that it does so without causing significant additional queuing at the bottleneck link.

V. Concluding Remarks

In this paper, we propose the notion that a congestion-control protocol can boldly probe for an exponentially-wide range of sending rates by limiting the impact of each probe. We realize this notion by: (i) relying on a rate-based transmission where the inter-packet spacing is carefully controlled to probe for a wide range of rates, and by (ii) relying on one-way delays for estimating whether a probing rate is larger than the avail-bw.

We use these ideas to design a novel protocol, RAPID, that exhibits two significantly desirable characteristics. First, it reduces by more than an order of magnitude the time it takes for a transfer to acquire spare bandwidth. Second, it maintains small and transient queues at bottleneck links and has a negligible impact on the performance of regular TCP transfers.

We are currently conducting large-scale evaluations to understand the sensitivity of RAPID to the values of its parameters, especially as network topologies and link capacities scale up. We are also designing mechanisms for ensuring intra-protocol fairness when several RAPID transfers co-exist.

References

[1] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zoharjan, “PCP: Efficient endpoint congestion control,” in *Proceedings of the 3rd Symposium on Networked Systems Design and Implementation (NSDI)*, May 2006.

[2] S. Floyd, “HighSpeed TCP for Large Congestion Windows,” RFC 3649 (Experimental), Dec. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3649.txt

[3] M. Fukuhara, F. Hirose, T. Hatano, H. Shigeno, and K. Okada, “SRF TCP: A TCP-friendly and fair congestion control method for high-speed networks,” in *Proceedings of the International Conference on Principles of Distributed Systems (OPODIS)*, ser. Lecture Notes in Computer Science, vol. 3544. Springer, 2005, pp. 169–183.

[4] Jin, Wei, and Low, “FAST TCP: Motivation, architecture, algorithms, performance,” in *Proceedings of IEEE INFOCOM*, vol. 4, March 2004, pp. 2490–2501.

[5] T. Kelly, “Scalable TCP: Improving performance in highspeed wide area networks,” in *First International Workshop on Protocols for Fast Long-distance Networks*, February 2003.

[6] I. Rhee, L. Xu, and S. Ha, “CUBIC for fast long-distance networks,” August 2007, Internet Draft.

[7] L. Xu, K. Harfoush, and I. Rhee, “Binary increase congestion control for fast, long distance networks,” in *Proceedings of IEEE INFOCOM*, March 2004.

[8] S. Floyd and M. Allman, “Specifying New Congestion Control Algorithms,” RFC 5033, 2007. [Online]. Available: http://www.ietf.org/rfc/rfc5033.txt

[9] V. Ribeiro, R. Riedi, R. Baraniuk, J. Navratil, and L. Cottrell, “pathChirp: Efficient available bandwidth estimation for network paths,” in *Passive and Active Measurement Workshop* , April 2003.

[10] M. Weigle, P. Adurthi, F. Hernandez-Campos, K. Jeffay, and F. Smith, “Tmix: A tool for generating realistic application workloads in ns-2,” *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 3, pp. 67–76, 2006.