Packet-Scale Congestion Control Paradigm

Rebecca Lovewell, Qianwen Yin, Tianrong Zhang, Jasleen Kaur, and Frank Donelson Smith

***Abstract*— This paper presents the packet-scale paradigm for designing end-to-end congestion control protocols for ultra-high speed networks. The paradigm discards the legacy framework of RTT-scale protocols, and instead builds upon two revolutionary foundations—that of continually probing for available bandwidth at short timescales, and that of adapting the data sending rate so as to avoid overloading the network. Through experimental evaluations with a prototype, we report high performance gains along several dimensions in high-speed networks—the steady-state throughput, adaptability to dynamic cross-traffic, RTT-fairness, and co-existence with the conventional TCP traffic mixes. The paradigm also opens up several issues that are less of a concern for traditional protocols—we summarize our approaches for addressing these.**

***Index Terms*— Communications technology, communication systems, protocols, transport protocols.**

I. Introduction

WHY *Ultra-High-Speed*[1] *Transport Protocols?* End-to-end data transfer rate requirements in the computational science communities as well as within large data centers involved in enterprise computing are soon to approach the *terabit-per-second* regime [1]–[3]. While high-speed network infrastructure is increasingly being deployed [4], [5], such capacity can not be made available on an end-to-end basis to applications if the underlying transport protocols do not scale correspondingly. One of the key components that determines the scalability of a transport protocol is its congestion control protocol—such a protocol should adaptively discover the end-to-end spare bandwidth available for data transfer in a quick and non-intrusive manner. In this paper, we consider the design of ultra-high-speed congestion control.

State of the Art: Not-so-High Speed Congestion Control: The design of high-speed congestion control protocols has been a fairly vibrant area of research over the last decade [6]–[14]. Existing designs focus mainly on two approaches for achieving scalability: (i) the use of more-aggressive-increase and milder-decrease factors as compared to the additive-increase-multiplicative-decrease of TCP NewReno; and (ii) the use of "early" congestion indicators (such as increase in end-to-end delays) in addition to packet loss [6], [10], [11]. While these designs have significantly improved upon the scalability of NewReno, even the best-performing designs struggle to achieve even 10Gbps of steady-state throughput per stream[2] *without* causing significant congestion (as observed both in wide-area experiments and in simulations [15]–[19]).

A New Paradigm: We argue that the state of the art offers only limited scalability because *all* of the high-speed designs retain the legacy design framework of *RTT-scale* protocol operations[3]—as explained in Section II, this framework fundamentally limits the ability of a protocol to operate at ultra-high speeds without nearly causing congestion collapse. Instead, we show that if this legacy mindset is discarded, it is possible to adopt a novel paradigm of *packet-scale congestion-control*, in which the protocol operates at a frequency close to the frequency of packet transmissions [22]. This paradigm allows the congestion-control timescale to be shrunk by several orders of magnitude over current protocols, especially in high-speed networks. This reduced timescale can then be exploited to *probe for a wide range of rates within an RTT without overloading* the network. This is the most distinguishing feature of the paradigm—existing "high-speed" protocols take orders of magnitude longer to probe for a similar range; and *no* existing protocol can do even that without overloading the network once it gets close to the available-bandwidth (henceforth, referred to as *avail-bw*). Additionally, discarding of the RTT-scale framework allows the paradigm to address as never before, two fundamental issues that have remained elusive to protocol designers—RTT fairness as well as friendly co-existence with conventional Internet TCP traffic.

Our experimental evaluation with a prototype of the paradigm illustrates that it has potential for high impact along several dimensions:

- *Speed/Overhead:* While most RTT-scale "high-speed" protocols struggle with the speed-overhead tradeoff, the packet-scale paradigm could allow a protocol to detect

Manuscript received August 14, 2014; revised June 19, 2015 and February 10, 2016; accepted May 27, 2016; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor Y. Ganjali. Date of publication July 26, 2016; date of current version February 14, 2017. This work was supported in - part by the National Science Foundation under Awards CNS--0347814, CNS--1018596, and OCI--1127413. *(Qianwen Yin and Tianrong Zhang contributed equally to this work.)*

R. Lovewell was with The University of North Carolina at Chapel Hill, Chapel Hill, NC 27599 USA. She is now with the Caktus Group, Durham, NC 27701 USA (e-mail: lovewell@cs.unc.edu).

Q. Yin, T. Zhang, J. Kaur, and F. D. Smith are with The University of North Carolina at Chapel Hill, Chapel Hill, NC 27599 USA (e-mail: qianwen@cs.unc.edu; trzhang@cs.unc.edu; jasleen@cs.unc.edu; smithfd@cs.unc.edu).

Digital Object Identifier 10.1109/TNET.2016.2591018

[1]In this paper, the term "ultra-high" refers to networks with capacity in the 10Gbps to multi-terabit regime.

[2]It is important to emphasize that we are referring to *single-stream* throughput here. Researchers have indeed demonstrated a much-higher aggregate throughput using multiple simultaneous streams (e.g., in [15]). However, it is gains in single-stream throughput that can truly benefit scientific applications and infrastructure without requiring changes to the application code-base.

[3]There are some protocols, especially those targetted for real-time streaming applications, that do not rely solely on an RTT-scale framework [20], [21]—however, to the best of our knowledge, *none* of these are designed for generic large data transfers, or have been evaluated on high-speed networks.

end-to-end avail-bw of up to *multi-terabits within a few RTTs*, while causing *negligible router queuing footprints!*

- *Adaptability:* The fine timescale at which the paradigm operates allows it to exploit efficiently short-timescale changes in end-to-end avail-bw—this is true while most existing protocols either do not achieve high utilization or are able to do so only by maintaining very large packet queues at the bottleneck router.
- *Incremental Deployability/TCP Co-existence:* Due to its low-queuing footprint, the packet-scale paradigm allows an ultra-high speed transfer to share a network with highly-multiplexed aggregates of conventional low-speed TCP transfers *without affecting the performance of the latter*. None of existing "high-speed" protocols have been able to achieve this property without sacrificing on their efficiency.
- *RTT-fairness:* By shedding the RTT-scale legacy framework of operation, the paradigm allows the design of end-to-end congestion-control that is *truly RTT-fair* and does not favor short-RTT transfers (again, unlike any RTT-scale high-speed protocol).

In the rest of this paper, we examine the RTT-scale framework of existing protocols in Section II. We present the packet-scale paradigm in Sections III and IV. We describe a prototype and present its experimental evaluation in Section V. In Section VI, we describe the open challenges that need to be addressed before the paradigm can be deployed. We conclude in Section VII. All of the experiments presented in this paper have been conducted using the ns-2 simulator [23]. While some experimental results are discussed in Section II, the experimental setup is described in detail only in Section V.

II. State of the Art: RTT-Scale Congestion Probing

While several aspects of a transport protocol and its configuration could influence the data transfer speeds achieved by applications, one of the most significant factors has been recognized to be its congestion-control protocol. For large data transfers, a congestion-control protocol constantly probes the network for the highest rate at which data can be transferred without causing network congestion. Nearly all existing end-to-end[4] congestion-control protocols do this by fundamentally operating at an RTT-long timescale[5]: (i) they probe for a candidate transfer rate by adopting it for an RTT-duration, and (ii) depending on the feedback received, they increase/decrease the probing rate adopted in the next RTT by some factor. Existing protocols differ mainly in two aspects—the increase/decrease factors used as well as the feedback metric used for detecting congestion. Research in designing high-speed congestion-control has focused on: (i) designing more aggressive combinations of increase/decrease factors [6]–[12], as well as (ii) relying on feedback metrics that provide "early" indications of congestion (such as increase in packet delays, in addition to packet losses) [6], [10], [11]. To date, however, *all* attempts at high-speed end-to-end congestion-control[6] have retained the *RTT-scale* design framework, which has been adopted (as a never-before re-examined legacy) from the early TCP designs of more than two decades ago. We argue below that this design framework limits the scalability of protocols due to an inherent speed-overhead tradeoff/dilemma.

The Speed-Overhead Dilemma: All end-to-end protocols need to "create" at least some degree of network congestion before they realize that their probing rate has exceeded the end-to-end avail-bw. Because of this property, an RTT-scale high-speed protocol can not afford to use very aggressive *increase factors* during Congestion Avoidance, for fear of severely overloading the network. To understand why, consider a hypothetical RTT-scale protocol that aggressively doubles its probing rate every RTT (which is fairly aggressive by current standards). Compared to existing "high-speed" protocols, the protocol would indeed be able to probe for a much wider range of rates within a few RTTs. But the problem occurs once the protocol has reached a probing rate close to, but slightly lower than, the current path avail-bw—the protocol is unaware of this and doubles its probing rate yet again in the following RTT, thereby significantly overshooting the avail-bw (by nearly 100%). With the RTT-scale framework, this situation can cause significant overload in the network—this is because the new higher rate is adopted for a complete *RTT-worth of duration*. Consequently, the data dumped in excess of what can be handled by the network $(= (probing\text{-}rate - avail\text{-}bw) * RTT)$ can be prohibitively large—in high-speed networks, this can amount to tens-of-thousands or even more of extra packets. To avoid such congestion-collapse-like conditions from occurring, *all* high-speed congestion-control designs rely on probing behavior that is not sufficiently aggressive for scaling to upcoming ultra-high speed networks.

In Fig 1 we plot the average link utilization achieved by a single large transfer in 1-40 Gbps networks, in the presence of a fiber error rate of 10^{-6}.[7] We find that while existing protocols perform well in 1 Gbps networks, most can barely utilize 10-40% of the available bandwidth at 10 Gbps and higher speeds—Fig 2 shows that the few that can achieve higher utilization, do so only at the cost of unreasonably large queue occupancies (note the log-scale of the y-axis). These simulation numbers match those reported even in wide-area evaluations of prototypes of these protocols [16]–[18].

[4]The literature also contains several *explicitly-guided* protocols that rely on explicit feedback from routers (that is not available in the current Internet) [24]–[28]—in this project, we consider only protocols that do not rely on such support and can be deployed in the Internet. In the rest of this document, "congestion-control" refers to only *end-to-end* designs.

[5]The round-trip time (RTT) of a transfer is the total time it takes for a data segment to transfer from the sender to the receiver, and for the corresponding acknowledgment to make its way back.

[6]Existing protocols that rely on RTT-scale framework include: (i) loss-based protocols ranging from NewReno variants [29] to HighSpeed [7], Scalable [8], CUBIC [9], [12]; (ii) delay-based protocols such as Vegas [30], FAST [6], Illinois [10], Compound-TCP [11]; and (iii) rate-based protocols such as in [31]–[33] (which smooth out protocol behavior and operate at even slower timescales). Even Westwood [34], PCP [35], UDT [14], RBUDP, and NF-TCP [36], which employ some form of short-scale probing every so often, have not exploited the full potential of the concept—to be discussed more in Section VII.

[7]The corresponding experiments were run on the ns-2 simulator, using a path with 100ms RTT, and with a packet size of 1040 B.

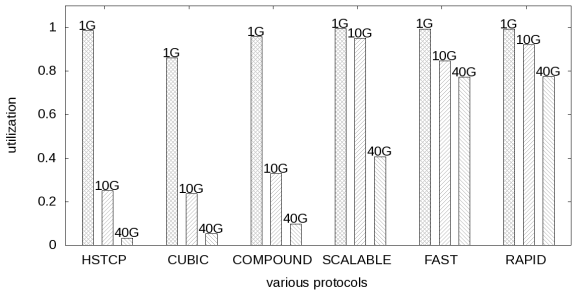


Fig. 1. Single-stream Utilization in 1G-40G networks (10^{-6} link error rate).

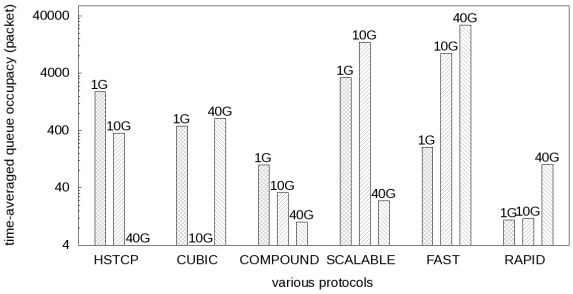


Fig. 2. Average Queue Occupancy: single stream in 1G-40G networks.

Other than limiting scalability, the RTT-scale framework also leads to the following two significant limitations.

Poor RTT-Fairness: The RTT-scale framework also leads to the undesirable property of *RTT-unfairness*. Since all transfers sharing a bottleneck link operate at the scale of their respective RTTs, flows with shorter RTTs have a higher rate-updating frequency—consequently, these are able to obtain an unfairly large share of the bottleneck bandwidth. While some recent protocols reduce the degree of unfairness by adopting RTT-aware increase-factors during Congestion Avoidance, to the best of our knowledge, there is no existing window-based protocol that is simultaneously efficient as well as truly RTT-fair [6], [11], [14], [37]. Figures 3(a)-(e) plot the throughput achieved by 3 transfers with RTTs ranging from 100ms to 200ms, when they use existing protocols—the transfers arrive and depart the network at different times. We find that *none* of these prominent protocols results in an equal allocation of throughput for co-existing transfers.

Poor Deployability Aspects: Due to the speed-overhead tradeoff, all recent high-speed designs are torn between the desire to aggressively utilize large spare bandwidth, while not degrading the performance of conventional low-speed TCP traffic mixes that share the same network queues [38]. To illustrate the problem, we generate an Internet-derived representative mix of TCP transfers[8] and aggregate these on a 1 Gbps shared bottleneck link. Fig 4(1a) plots the spare bottleneck bandwidth left in the network by this aggregate—Tmix validations have shown that this profile is fairly representative of production Internet links [39]. We then transfer a single large file through the bottleneck link using several prominent protocols. Figs 4(2a)-(6b) plot (a) the throughput achieved by the single large transfer, and (b) the bottleneck queue occupancy. We find that these protocols can be fairly intrusive in occupying the shared bottleneck queues, and their throughput can significantly over-shoot the spare bandwidth in the network—the amount of bottleneck bandwidth available to low-speed TCP transfers may be reduced by up to 50%)! Fig 5 plots the distribution of the response times observed by the low-speed TCP transfers present within the traffic mix. We find that, for the majority of the "mice" transfers, the average response time increases by 50-100 ms. Recent studies suggest that such increases can significantly impact user retention and revenues for modern web-services [40]!

Once again, the RTT-scale framework limits the ability of a protocol to address this issue. This is because both high-speed protocols and low-speed TCP operate at the RTT-scale—the only difference between the protocols lies in the aggressiveness with which they increase their probing rates in successive RTTs. Since a high-speed protocol is more aggressive, it will quickly occupy spare bandwidth (and more) before a low-speed TCP transfer gets a chance to do so. While protocols like FAST that additionally rely on delays as an indicator of congestion, can be configured not to be aggressive to TCP, they may not be able to efficiently utilize dynamically-varying avail-bw, as is the case on production Internet links.

We next present the novel packet-scale paradigm that allows us to rethink congestion-control designs that operate at considerably smaller (and closer-to-optimal) timescales, and helps in overcoming each of the above limitations.

III. The Packet-Scale Paradigm: Key Concepts

The packet-scale congestion control paradigm discards the RTT-scale *frequency* and *duration*[9] of probing for a candidate transfer rate, which is characteristic of most existing protocols. Instead, it introduces two novel and transformative concepts:

Fine-Scale Probing: It uses probing timescales that are smaller by several orders of magnitude than existing protocols (especially for transfers on long Internet paths). Specifically, rather then sending packets at a given probing rate for an RTT-worth of duration, this paradigm *probes for a target rate only for much smaller durations*. It does so by sending only a much smaller number of packets with carefully controlled inter-packet gaps set according to the desired probing rate. This feature has immediate consequences for the speed-overhead tradeoff. First, the frequency with which different rates are probed for, increases from once-per-RTT to numerous-per-RTT (potentially tens or hundreds of rates)—thus, the speed with which a wide range of candidate rates can be probed for can be orders of magnitude higher in this paradigm. Second, shrinking of the probing timescale also shrinks the total amount of extra packets introduced in the network when the probing rate exceeds the network avail-bw—thus, a protocol can adopt fairly aggressive increase factors for selecting successive probing rates during Congestion Avoidance, without causing significant congestion.

[8]We use the Tmix [39] traffic generator for this experiment. Tmix has been shown to reproduce traffic characteristics at fine timescales, as are observed on production Internet links. More details are included in Section V-E.

[9]It is important to clarify that the path RTT for a transfer represents the *minimum feedback delay*—the sender can not get feedback on a probing rate it has adopted before this time has elapsed. The packet-scale paradigm, instead removes the dependence of the *frequency and duration of probing* on the RTT.

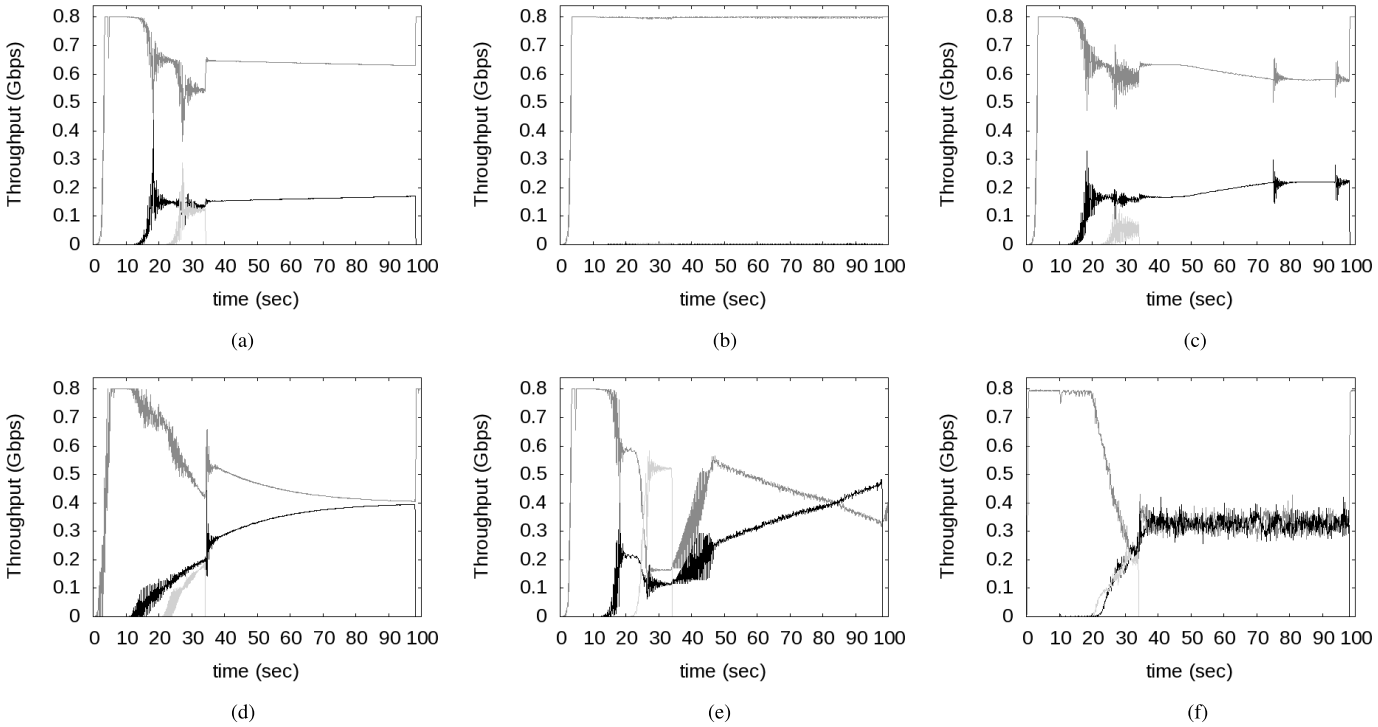


Fig. 3. Intra-protocol Fairness (flow 1 (100 ms RTT): 0-100 s, flow 2 (200 ms RTT): 10-98 s, flow 3 (150 ms RTT): 20-34 s). (a) HighSpeed. (b) Scalable. (c) CUBIC. (d) FAST. (e) Compound. (f) RAPID.

This framework evaluates the suitability of each probing rate used by checking if the corresponding packets increased the queue-buildup at the bottleneck link—this would be evident when the inter-packet gaps consistently show an increasing trend from their pre-set values by the time the packets reach the receiver. This concept is based on the principle of *self-congestion*, which has been used widely in the design of bandwidth-estimation tools [41], [42].

Probing-Without-Overloading: Fine-scale probing reduces significantly the overload caused when the probing rate exceeds the network avail-bw. However, it does not eliminate it—this overload can still add up to be a big concern in highly-multiplexed ultra-high speed networks. This issue is explicitly addressed by the probing-without-overloading feature, in which the paradigm attempts to *probe for higher transfer rates without overloading* the network at reasonably small sub-RTT timescales. This is done by sending successive packets in short-length groups (*p-streams*), such that: (i) the *average* rate of packets in a p-stream is no more than the most recently discovered estimate of the end-to-end avail-bw (no overloading at p-stream timescale), but (ii) some packets within a p-stream are sent at rates that can be much larger that this estimate (high-rate probing at sub-p-stream timescales). Thus, while high rates used at sub-p-stream timescales can cause small transient queues, the queues drain out at p-stream (and larger) timescales. This is a unique feature of the paradigm—no existing end-to-end protocol probes for higher rates without causing some degree of congestion at RTT-and-higher timescales.

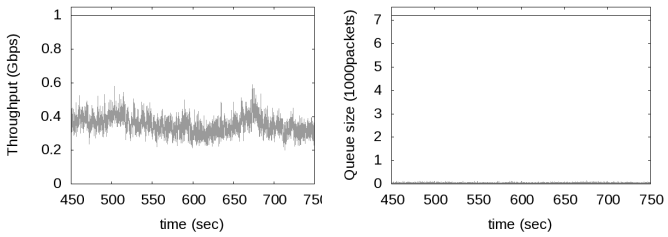
Furthermore, in order to maintain the average rate at p-stream timescales, the paradigm sends packets within the p-stream at rates both larger as well as *smaller* than the most recent estimate of the avail-bw—each p-stream, consequently, simultaneously probes for the possibility that the avail-bw might have increased *or even decreased* from the most-recent estimate. This additional unique feature of the paradigm gives it excellent agility in adapting to small-scale changes (whether increase or decrease) in end-to-end avail-bw.

It is important to note that probing-without-overloading can perhaps be attempted even in RTT-scale protocols—for instance, by adopting low and high probing rates in alternating RTT intervals. However, probing at high rates for even an RTT-worth of duration would still cause prohibitive congestion in ultra-high speed networks (especially if the high-probing phases of multiple transfers overlap). The fine probing timescales of the packet-scale paradigm help avoid this issue.

Buy Two, Get Four!: As detailed in Sections IV and V, by shedding the RTT-scale framework, the paradigm is also free from RTT-unfairness. Additionally, the combination of fine-scale probing and probing-without-overloading allow ultra-high-speed transfers using the paradigm to share the network with conventional TCP traffic without impacting the performance of the latter. Both of these issues have remained elusive to existing protocols.

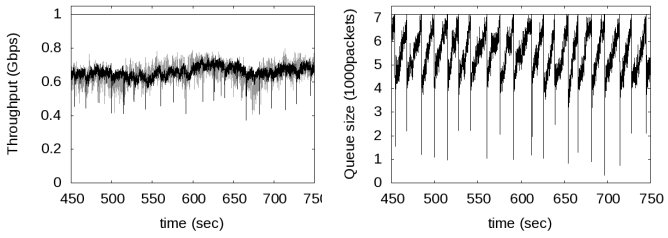
IV. Realizing the Paradigm: Mechanism Design

The packet-scale paradigm introduces the concept of *p-streams*, in which: (i) packets are transmitted by the sender with controlled probing rates; (ii) the changes in the inter-packet gaps are used to estimate the end-to-end avail-bw; and (iii) the bandwidth estimates are used to shape future p-streams in a probing-without-overloading manner. Below, we present mechanisms for realizing each of the above.



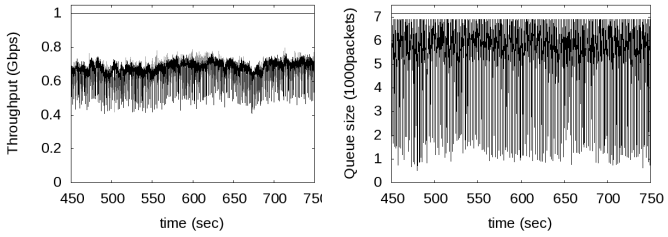
(1a) Tmix-only Throughput

(1b) Tmix-only Buffer



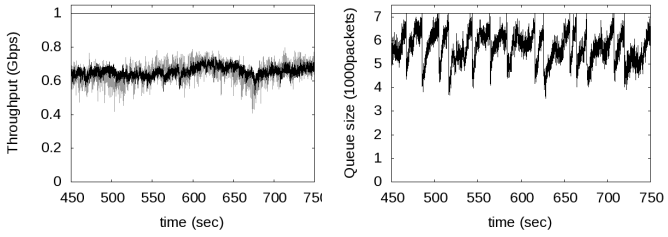
(2a) HSTCP Throughput

(2b) Tmix+HSTCP Buffer



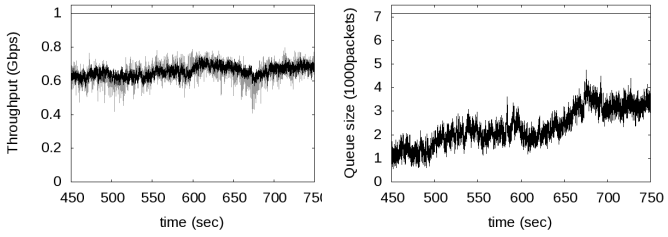
(3a) STCP Throughput

(3b) Tmix+STCP Buffer



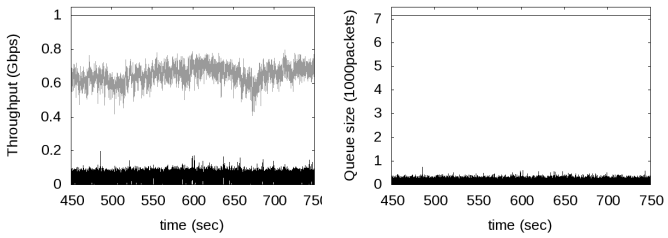
(4a) CUBIC Throughput

(4b) Tmix+CUBIC Buffer



(5a) Compound Throughput

(5b) Tmix+Compound Buffer



(6a) FAST Throughput

(6b) Tmix+FAST Buffer

Fig. 4. Performance with Tmix Cross Traffic.

A. Multi-Rate Based Packet Transmission at the Sender

Most existing congestion-control protocols rely on sliding-window based packet transmissions, in which the sender sends as many data packets as allowed by the congestion-window, and then waits for acknowledgements to arrive. This results in "ack-clocked" packet transmissions, in which each arriving ack triggers the transmission of new data packets.

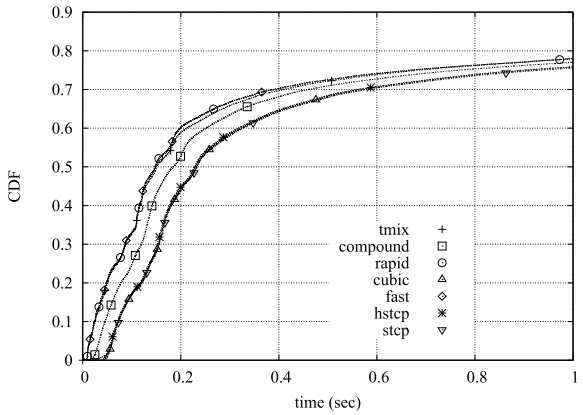


Fig. 5. Impact of high-speed transfer on TMIX Response Times.

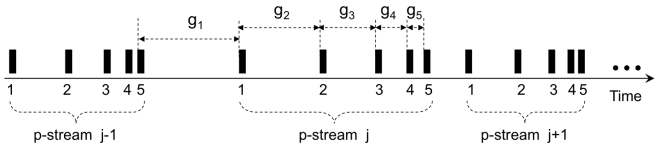


Fig. 6. Illustration of a p-stream ($g_i = \frac{P_i}{r_i}$).

In contrast, a packet-scale sender continuously[10] transmits data in logical groups, referred to as *multi-rate probe-streams (p-streams)*, in which packets are sent according to a pre-determined rate. Specifically, the i^{th} packet in a p-stream is sent at a rate of r_i—this is achieved by ensuring that the times at which packets i and $i-1$ are sent differ by $\frac{P_i}{r_i}$, where P_i is the size of packet i (see Fig 6).[11] For all $i > 1$, $r_i \geq r_{i-1}$.[12]

The sender explicitly controls the average sending rate of a p-stream, which is given by:

$$r_{avg} = \frac{P_1 + P_2 + \cdot + P_N}{\frac{P_1}{r_1} + \frac{P_2}{r_2} + \cdot + \frac{P_N}{r_N}} \tag{1}$$

where N is the number of packets in a p-stream. When all packets have the same size, we get: $r_{avg} = \frac{N}{\sum_{i=1}^{N} 1/r_i}$.

B. Estimation of End-to-End AB

End-to-end congestion control protocols estimate the avail-bw by searching for the largest sending rate that does *not* cause congestion. Existing protocols detect congestion by looking for signs of high bottleneck buffer occupancy—indicated by either an inflated RTT [6], [10], [11], and/or by encountering packet losses [6]–[12].

[10]As described in [43], if multiple probe-rate tools share network paths with the same bottleneck router they will collectively overestimate the avail-bw on the bottleneck *if* the probe streams are not sent out continuously, but instead intermittently. Therefore, all probing should be done in a continuous manner—even if it means sending shorter p-streams when sufficient packets are not available to be sent.

[11]The rate for the 1^{st} packet is achieved by setting the gap between it and the last packet of the previous p-stream to $\frac{P_1}{r_1}$

[12]It is also possible to base bandwidth-estimation on a p-stream model in which packet rates are non-increasing within a p-stream [44].

The packet-scale paradigm instead only estimates if the probing rate is larger than the avail-bw left over by *simultaneously-arriving* cross-traffic at the bottleneck buffer—it does not attempt to measure the bottleneck buffer occupancy (which could have been built-up due to traffic that arrived previously). Specifically, if the first byte of the i^{th} packet of a p-stream *arrives* at the bottleneck buffer at time t_i, then the paradigm attempts to measure: $AB(t_0, t_N) = C - B(t_0, t_N)$, where t_0 is the time at which the last packet of the previous p-stream arrived, $B(t_0, t_N)$ is the total amount of cross-traffic that arrives during $[t_0, t_N)$, and C is the transmission capacity of the bottleneck link.

To measure $AB(t_0, t_N)$, the paradigm relies on the principle of *self-induced congestion* that has been used extensively in the bandwidth-estimation literature [41], [42]. Intuitively, if q_i is the queuing delay experienced at a bottleneck link by the i^{th} packet of a p-stream, and if AB represents the (constant) avail-bw encountered by the p-stream, then:

$$q_i \leq q_{i-1}, \quad \text{if } r_i \leq AB \tag{2}$$

$$q_i > q_{i-1}, \quad \text{otherwise} \tag{3}$$

Thus, 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 (since $r_i > r_{i-1}$, for all $i > 1$)—due to this "self-congestion", each of these packets will experience a larger one-way delay (and a larger increase in the pre-set inter-packet gap) than its predecessor. Thus, the smallest rate r_{i^*-1} at which the receiver observes an increasing trend in the inter-packet gaps can be used to compute an estimate of the current avail-bw as: $ABest = r_{i^*-1}$.[13] The actual analysis uses several heuristics to account for bursty cross-traffic—we refer the reader to [45] for details and the precise formulation. In addition, transient queuing can occur on multiple occasions as packets traverse shared resources, and can introduce (sometimes significant) noise in inter-packet gaps—in Section VI, we describe our parallel work on handling such noise.

C. Probing-Without-Overloading

When the sender computes a new $ABest$ value, it updates the r_{avg} of the next p-stream as: $r_{avg} = \mathcal{F}(ABest)$, where $\mathcal{F}(.)$ is a smoothing function. Thus, the transfer acquires an *average* sending rate that closely follows the smoothened avail-bw within an RTT. The sender then selects an appropriate set of rates, $r_1, \ldots, r_N$, for the next p-stream such that the average of these is equal to r_{avg}, as computed in Eqn (1).

The above mechanism helps simultaneously achieve two desirable properties. First, by setting r_{avg} equal to the (smoothened) estimated avail-bw, the sender helps to ensure that the average load on the bottleneck link does not exceed its capacity—this is crucial for maintaining small and transient queues at the bottleneck links. Second, by selecting a set of rates which includes values larger (e.g., r_N) as well as smaller (e.g., r_1) than r_{avg}, a p-stream is able to simultaneously probe for both increase and decrease in the current end-to-end avail-bw—this greatly helps the sender in quickly detecting and adapting to changes in the avail-bw.

The use of probing-without-overloading is expected to significantly reduce the number of congestion-related losses that a sender experiences. However, losses may still occur. Like in NewReno, packet losses are detected and recovered from using either the retransmission timeout or the fast retransmit/recovery mechanisms. Bandwidth probing is turned off during loss recovery. After a retransmission timeout, the sender reduces its sending rate to the initial sending rate, r_{init}, and begins the bandwidth search process afresh. After loss recovery with fast retransmit/recovery, the sender reduces r_{avg} by a multiple β and resumes the sending of p-streams.

D. Smoothing for Fairness and Robustness

We next consider two desirable properties of a congestion-control protocol—the first of these is *robustness to dynamic network conditions*. The packet-scale paradigm has excellent adaptability to dynamically-varying avail-bw; but its short p-streams could also be sensitive to transient, short-scale traffic burstiness, especially at the timescales typically encountered on high-bandwidth, highly-multiplexed links. The packet-scale paradigm filters out the effect of such fine-scale transient queuing at network links by using the smoothing function, $\mathcal{F}(.)$—such a function dampens out sudden changes in $ABest$ values that are caused by bursty cross-traffic.

The second property we consider is that of intra-protocol fairness, which characterizes the ability of a protocol to result in a fair allocation of bottleneck bandwidth among co-existing transfers. For the packet-scale paradigm, this boils down to the question: *how do the p-streams of different transfers interact—how can transfers obtain a fair share of the avail-bw?* In this section, we address two sources of unfairness that have been identified in the literature [46], [47].

1) RTT-Fairness: Most window-based congestion control protocols have been shown, both experimentally and analytically, to suffer from *RTT-unfairness*—transfers with a long RTT get a lower throughput than short-RTT transfers [6], [46], [47]. This happens because window-based protocols update their sending rates *once per RTT*—in heterogeneous RTT environments, this RTT-dependence results in differences in the rate updating frequency as well as rate increments, and results in a bias against long RTT transfers [47].

Fortunately, the p-stream-based design of RAPID is not influenced by the value of RTT[14]—a RAPID sender continuously send p-streams, for both long and short RTT transfers. The rate-updating frequency (once per p-stream) as well as rate updating amount (determined by $ABest$) are independent of the RTT—consequently, and by design, *the packet-scale paradigm truly does not suffer from RTT-unfairness.* Our experiments with a prototype in Section V confirm this.

[13] If no increasing trend is detected in a p-stream, r_N is taken as the AB estimate. Also, if the increasing trends starts at the first packet ($i^* \leq 1$), $\frac{r_1}{2}$ is returned as the AB estimate.

[14] As noted before—for *any* congestion control protocol, the RTT represents a *minimum* feedback delay; a sender can not get end-to-end feedback on a probing rate that it has adopted before this amount of time has elapsed. What the packet-scale paradigm instead does is that it removes RTT-dependency from the *frequency* and *duration* of probing.

2) Flow-Rate Fairness: The packet-scale sender computes avail-bw estimates and updates its sending rate once per p-stream. Depending on the length of theirs p-streams, two competing flows may not get to do this with the same frequency. Even if the two flows send equally-long p-streams, their avail-bw estimates will depend on several factors, including their average sending rates r_{avg}, as well as the offsets between when their p-streams arrive at the bottleneck queue [43]. In general, this implies that when a low-rate transfer is competing with a transfer that has already attained a high sending rate, it will need additional mechanisms to ensure that it achieves a fair share of throughput.

The packet-scale paradigm achieves this, again, by means of the smoothing function, $\mathcal{F}(.)$, which dampens out changes in the $ABest$ values—in the packet-scale paradigm, the function is additionally designed to give a relative advantage to transfers with lower average sending rates (smaller r_{avg}), against those with higher rates. Specifically, the smoothing function ensures that when the $ABest$ values estimated by a flow increase (or decrease), the change is dampened/smoothened by a factor that is inversely (or directly) related to r_{avg}—consequently, flows with a lower r_{avg} acquire spare bandwidth more aggressively, and give up bandwidth less aggressively. Section V describes the choice of $\mathcal{F}(.)$ for one prototype of the paradigm.

E. Co-Existence With Conventional Internet Traffic

By design, the packet-scale paradigm is expected to be quite non-intrusive to regular low-speed TCP NewReno transfers. The prime reasons for this are that: (i) it relies on increased queuing delays for detecting congestion, whereas TCP reduces its sending rate only on witnessing packet losses (that occur only once the buffers fill up and overflow); and (ii) the paradigm receives congestion-signals at a much higher frequency (once per p-stream) than TCP. When a router carrying both TCP and packet-scale transfers gets congested, the latter would respond to the congestion (and reduce their sending rates) much earlier than the TCP transfers would. This would ensure that the performance of the low-speed TCP transfers is not significantly impacted due to the presence of high-speed packet-scale transfers.[15] The downside is that in the presence of long-lived TCP transfers, packet-scale transfers would obtain lower throughput than the former. However, this problem plagues any network that simultaneously runs fundamentally different congestion-control protocols [6]—a simple solution is to provision routers with separate queues for traffic from different protocols.

F. Prototype Design Considerations

Within the packet-scale framework, there are several dimensions along which a prototype could make design choices. We discuss these below.

1) Setting $[r_1, \ldots, r_{N-1}]$: For given values of r_{avg} and N, there are infinite choices for the set of rates $[r_1, \ldots, r_{N-1}]$ that satisfy Equation 1. However, the larger is the range $[r_1, r_N]$, the faster would be the AB-search process—this is because a single p-stream would now probe for a wider range of sending rates. For a given N, on the other hand, the larger is this range, the coarser would be the granularity of probing rates used (and the avail-bw estimates obtained). Thus, while probing rates that grow multiplicatively within a p-stream ($r_i = m.r_{i-1}$) would yield a better range than rates that grow additively, the latter would yield fine-grained/high-precision estimates.

The length of a p-stream is also faced by opposing considerations. A larger value of N would improve the AB-search range ($[r_1, r_N]$); however, a larger p-stream would also be more intrusive to cross-traffic (by sending more packets at rates larger than r_{avg}).

2) Speeding Up a Slow Start: The packet-scale paradigm faces a dilemma similar to all congestion-control protocols—how to obtain the initial $ABest$ (or the initial r_{avg}) for a new transfer? The main concerns here are that the initial r_{avg} should neither be too high (larger than the avail-bw) for a given network path, nor should it be too low for quickly estimating and ramping up to the avail-bw. A packet-scale sender can address this challenge in a manner similar to all existing protocols—by defining two phases of protocol operation, *slow-start* and *congestion-avoidance*. The goal of the slow-start phase would be to quickly obtain a coarse estimate of the avail-bw; fine-grained refinements can then be made in the congestion-avoidance phase. Such a distinction between the two phases can be implemented by selecting a much wider (and coarse-grained) range of probing rates $[r_1, \ldots, r_{N-1}]$ in the slow-start phase, and switching to a narrower and fine-grained search range in congestion-avoidance.

3) Fixed-N vs. Fixed-L: There are at least two ways in which a packet-scale sender can configure the length of its p-streams—each p-stream can either consist of a constant number of bytes (fixed-N) or can be of a fixed length in time (fixed L), where $L = \sum_{i=1}^{N} \frac{P_i}{r_i}$. Again, the choice is guided by opposing considerations. On the one hand, fixing N can create p-streams that probe at extremely small timescales in high-bandwidth environments (and are, thus, prone to getting impacted by noise). On the other hand, fixing L can result in p-streams that carry large volumes of data in high-bandwidth environments (by sending more packets at rates larger than r_{avg}, these are more intrusive to cross-traffic).[16]

4) The Smoothing Function, $\mathcal{F}(.)$: $\mathcal{F}(.)$ is used to achieve both robustness to burstiness as well as intra-protocol fairness. The "degree of smoothing" employed by such a function is guided by opposing considerations as well. While more aggressive smoothing helps achieve greater robustness and provides a greater opportunity for achieving fairness, it also reduces the responsiveness of the paradigm in adapting quickly to bandwidth changes in dynamic network environments.

[15] It is important to note that some delay-based high-speed protocols that rely on the RTT-scale framework also react to increased queuing delays [6], [11]. However, both the high-speed protocols and low-speed TCP operate at the RTT-scale—the only difference between the protocols lies in the aggressiveness with which they probe for and acquire additional bandwidth. Since a high-speed protocol is typically more aggressive, it will quickly occupy spare bandwidth (and more) before a low-speed transfer gets a chance to do so. We have conducted experiments with FAST under different combinations of parameter settings and find that its performance is either as mild as in Fig 4, or is as aggressive as the other protocols.

[16] When L is as large as the RTT of a transfer, the sender get closer to facing the speed-overhead dilemma in the same way as RTT-scale protocols.

TABLE I
DEFAULT RAPID PARAMETERS

Param	Default Value	Description
τ	0.250 s	smoothing parameter when $ABest > r_{avg}$
η	1.5	smoothing parameter when $ABest < r_{avg}$
N	90 packets	number of packets in a p-stream
m	1.039	spread factor
P	1000	maximum packet size
β	0.5	reduction factor after loss recovery
r_{init}	1 Mbps	initial r_{avg} in slow-start
N_{SS}	20 packets	p-stream size during slow-start
m_{SS}	2	spread factor during slow-start

V. PROTOTYPE & EXPERIMENTAL EVALUATION

In this section, we present the design and experimental evaluation of RAPID, a proof-of-concept prototype based on the packet-scale paradigm [19].[17]

A. RAPID: A Prototype

1) Probing Rates: A RAPID sender uses fixed-N p-streams, in which the probing rates follow a multiplicative relation as in:

$$r_i = m^{i-1} * r_1, \quad 1 \leq i \leq N \tag{4}$$

where m is a constant referred to as the spreadfactor [42]. The parameters m and N control the range and granularity of the probing rates, as well as the p-stream length—as discussed in Section IV-F.1, several considerations guide the configuration of these. Table I summarizes the default settings for all parameters used in RAPID.

2) $\mathcal{F}(.)$: The RAPID sender feeds the $ABest$ values through a set of exponentially-weighted moving-average (EWMA) filters as follows. Just before sending out a new p-stream, if the most-recently computed value of $ABest$ is larger than the current r_{avg}, it updates r_{avg} as:

$$r_{avg} = r_{avg} + \frac{L}{\tau}(ABest - r_{avg}) \tag{5}$$

where L is the duration of the new p-stream to be sent and τ is a time-based constant—τ represents (roughly) the time units it would take for r_{avg} to converge to an updated (higher) value of $ABest$. If, instead, the most-recently estimated value of $ABest$ is smaller than the current r_{avg}, it updates r_{avg} as:

$$r_{avg} = r_{avg} + \frac{1}{\eta}(ABest - r_{avg}) \tag{6}$$

where η is a constant—it represents (roughly) the number of p-streams it would take for r_{avg} to converge to an updated (lower) value of $ABest$.

It is important to note that, as intended, both of these filters create a relative advantage for low-rate transfers (which have longer-duration p-streams than high-rate transfers)—when avail-bw increases, the factor $\frac{L}{\tau}$ is larger for low-rate transfers; when avail-bw decreases, it takes longer for a low-rate transfer to send η p-streams (and reduce its sending rate). As discussed in Section IV-D, such a bias is needed for achieving intra-protocol fairness. The use of the EWMA filters also helps smooth out fine-scale transient burstiness.

3) Slow-Start: The RAPID sender adopts smaller p-streams and a more aggressive spreadfactor in slow-start. The slow-start phase ends with the first p-stream that yields an $ABest$ value that is less than the maximum probing rate (r_N) used by that p-stream.

The packet-scale design raises at least three types of concerns: *does fine-scale probing really help in accurately estimating avail-bw, especially in high-speed and dynamic-bandwidth environments? How do the p-streams of co-existing transfers interact—do the flows get a fair share of the avail-bw?* The paradigm seems to explicitly create short-scale burstiness at sub-p-stream timescales—*is it friendly to router queues and competing low-speed TCP traffic mixes?* In what follows, we experimentally study these issues using the RAPID prototype. It is important to note that this prototype is not meant to represent an optimal choice within the packet-scale design space, but is merely a proof-of-concept used to experimentally illustrate the benefits of the paradigm—indeed, we do expect that further research will yield better choices.

B. Experimental Setup

We use the ns-2 simulator for our evaluations. We have implemented RAPID in ns-2, reusing the NewReno code base for handling packet loss detection and recovery. Except where noted otherwise, we use version 2.34 of ns. We also evaluate CUBIC, HSTCP, STCP, Compound, and FAST using publicly-available ns-2 implementations [48]. The first three protocols are loss-based with fairly different window growth functions, while the last two are additionally responsive to increases in end-to-end delays. We use the default ns-2 parameters for most protocols.[18]

In our experiments we use a simple dumbbell topology in which multiple sources are aggregated at a single bottleneck link ($R_1 - R_2$ in Figure 7). The bottleneck link capacity is set by default to 1 Gbps, but is enlarged in some experiments as noted in their descriptions. All links other than the bottleneck have a transmission capacity of twice the bottleneck speed. All buffers are provisioned with a bandwidth-delay product (BDP) of buffers, where the delay is the average round trip time for transfers (specified for each experiment) and the bandwidth is that of the bottleneck link.[19] Unless mentioned otherwise, the link-layer frame size is set to 1000 B in all experiments.

The main performance metrics we study are: (i) the throughput obtained by transfers between sender and receiver nodes, (ii) the bottleneck link utilization, and (iii) the queue build-up at the bottleneck link. We sample each of these quantities at

[17]The RAPID protocol has undergone some changes from the version in [19]—the most noteworthy of these are the introduction of η and the influence of terminating excursions in the AB-estimation analysis. Details can be found in [45].

[18]For FAST, we use $\alpha = 200$ and $\beta = 200$—the default parameters resulted in fairly poor performance overall.

[19]Experiments with smaller buffer sizes are included in [45].

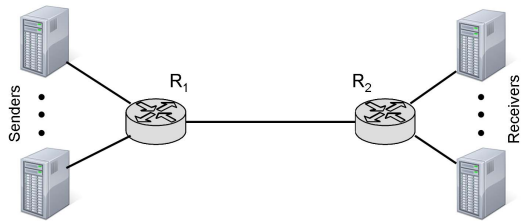


Fig. 7. Experimental Topology.

regular intervals of $50ms$ each. In what follows, we summarize our experiments and observations.

C. Steady-State Throughput in High-Speed Networks

We first evaluate the ability of the packet-scale paradigm to scale up to higher networks speeds. While only up to 10 Gbps of transmission speed is offered by widely-available commercial NICs today, ns-2 allows us to experiment with higher speeds as well. In order to study the scalability of transport protocols, we conduct several experiments across which we: (i) simulate different topologies, across which we vary the transmission capacity of the link $R_1 - R_2$ from 1 Gbps to 40 Gbps,[20] (ii) instantiate a single large transfer between a sender and a receiver, and (iii) simulate different transport protocols that control the sending of data.

Figs 1 and 2 plot, respectively, the average throughput achieved and the time-averaged queue occupancy for each of the high-speed protocols considered. As noted before in Section II, most existing protocols achieve only 10-40% utilization in 10 Gbps and higher speed networks. The only exception is FAST—Fig 2 illustrates, however, that even this RTT-scale protocol maintains significantly large network queues. RAPID, on the other hand, is able to scale well to ultra-high speeds, without creating a noticeable queue footprint. Thus, the small probing timescale, coupled with the concept of probing-without-overloading, helps the packet-scale paradigm in significantly alleviating the speed-vs-overhead tradeoff.

D. Intra-Protocol Fairness

We next evaluate the ability of the packet-scale paradigm to fairly allocate the bottleneck available bandwidth among multiple competing transfers. We are specifically interested in dynamic environments with heterogeneous RTTs—as noted in Section II, RTT-scale protocols are infamous for allocating lower throughput to transfers with larger end-to-end RTTs. The experiments we present in this section are inspired by an experiment conducted in [6] (in which FAST was shown to achieve better fairness than HSTCP and BIC).

We simulate a topology in which the transmission capacity of the link $R_1 - R_2$ is set to 800 Mbps. We introduce three high-speed TCP transfers, each with a different RTT, according to the following schedule: flow 1 (with 100 ms RTT) lasts from 0 - 100 s, flow 2 (with 200 ms RTT) lasts from 10 - 98 s, and flow 3 (with 150 ms RTT) lasts from 20 - 34 s.

[20]40 Gbps ns-2 simulations need massive amounts of memory and may take several days to complete even on high-end multi-core servers.

We run the experiment multiple times, each time using a different underlying congestion-control protocol. Fig 3 plots the throughput attained by the three transfers, with different underlying protocols. The black, grey, and light grey lines represent the throughput of the first, second, and third flow, respectively. We find that:

- All existing protocols that maintain large persistent bottleneck queues and use only packet losses as a prime congestion indicator, allocate bandwidth unfairly among the competing flows. The first flow, which has the smallest RTT, dominates over the other two later-arriving transfers (sometimes shutting them out completely, as is the case with Scalable)—as noted earlier, this is because transfers with shorter RTTs learn about (and react to) spare bandwidth earlier, gaining an unfair advantage over transfers with larger RTTs.
- After 34 s, there are only two remaining transfers that last till nearly the end of the experiment. We find that there is a great disparity in the throughput achieved with loss-based protocols. Fig 3 suggests that CUBIC and HIGHSPEED seem to be slowly reducing the gap between the two remaining transfers after 34 s—we, however, estimate using extrapolation that it would take more than 1500 seconds to close the gap.
- FAST, which is a delay-based congestion-control protocol, also exhibits unfairness for a long period. It takes more than 60 seconds to close the 300 Mbps throughput gap between the two flows.
- RAPID allocates bandwidth fairly among co-existing transfers, irrespective of their RTTs. Even when the first transfer has already acquired high throughput, the freshly-arriving second transfer is able to gain a fair share. This indicates that that choice of the filter $\mathcal{F}(.)$ in Rapid works well in allowing low-rate transfers to acquire a fair bandwidth share even in the presence of pre-existing high-rate transfers.

It is worth mentioning that while Rapid does achieve fair allocation of throughput, it also seems to lower the overall bottleneck utilization when multiple transfers co-exist. For instance, in Fig 3, the aggregate utilization falls down to 87% during the interval (30-35 s) in which all three transfers share bandwidth fairly. In Section V-G, we discuss the issue of parameter configuration, which helps balance this tradeoff between fairness and utilization in RAPID.

E. Efficiency in Dynamic Bandwidth Environments

Next, we simultaneously evaluate two important properties related to dynamic cross-traffic encountered by a high-speed protocol—first, how efficiently can RAPID work in the presence of representative, dynamic, and bursty cross-traffic? This boils down to asking: (i) is RAPID able to estimate bandwidth reliably even with bursty cross-traffic that introduces noise at fine timescales, and (ii) is RAPID able to adapt in an agile manner when the aggregate volume of competing cross-traffic changes dynamically? The second property we study is—to what extent does the presence of the high-throughput RAPID transfer adversely impact the performance of conventional TCP traffic that it co-exists with.

In order to study these two aspects, we use the publicly-available Tmix traffic generator for ns-2.35, which generates an empirically-derived mix of TCP flows that is representative of the mice-and-elephant TCP traffic aggregates observed on production Internet links [39], [49]—these aggregates include a majority of short transfers and some long transfers. Tmix uses real-world header traces of TCP traffic and derives from them the application-level socket-writing behavior for each TCP flow—it then reproduces in ns-2, the inter-connection arrival times, as well as the socket-level data generation pattern. TCP NewReno (ns-2 implementation) is used to transport the Tmix TCP data.

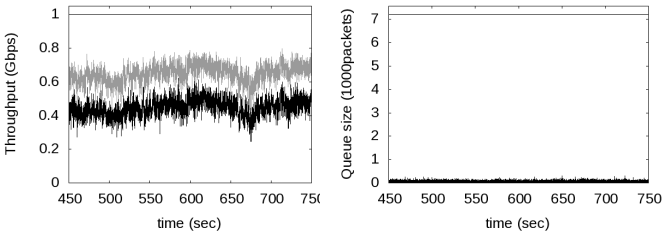
We simulate a topology in which the transmission capacity of $R_1 - R_2$ is set to 1 Gbps, and the Tmix sources and sinks clouds are attached to R_1 and R_2. We use a trace collected on a campus access link to generate a corresponding Tmix TCP traffic for a duration of 40 simulated minutes. Th traffic mix is typical of that observed widely on the Internet (majority of mice connections, but a few heavy elephants). All statistics presented here are collected from (roughly) the middle 15 minutes of the experiments (to exclude the ramp-up and ramp-down behavior of the traffic generator). All queues are allocated a BDP-worth of buffers (based on the RTT of the single high-speed transfer).

In the first experiment, we simulate (only) the Tmix traffic mix and observe the aggregate Tmix throughput (in consecutive 50 ms intervals) and queue size (sampled periodically every 50 ms) at the bottleneck router. In Fig 4(1a), we plot the spare bandwidth at the bottleneck (computed as the difference between 1 Gbps and the corresponding Tmix throughput); Fig 4(1b) plots the sampled buffer occupancy. It can be seen that even though the bottleneck queues are small, the traffic is fairly bursty and varies significantly around the average.[21]

We next conduct several experiments, in each of which, we use a different high-speed congestion-control protocol to additionally transfer, with a 60 ms RTT, a single large file through the bottleneck link. Figs 4(2a-6a) plot the throughput achieved by the single large transfer, and Figs 4(2b-6b) plot the bottleneck queue occupancy observed in each experiment. Figs 8(a-b) plot the same quantities when RAPID is used for the large transfer. We find that:

- When loss-based congestion-control protocols (including Compound, that has a predominantly loss-based reaction to congestion) carry the large high-speed transfer, they maintain high router buffer occupancy, and cause significant packet losses. Consequently, they are able to achieve a high throughput, albeit at the cause of causing significant queuing delays and congestion.
  It is important to note that the coexisting low-speed TCP NewReno transfers in the Tmix aggregate also suffer packet losses in the presence of an aggressive high-speed transfer. These low-speed transfers, consequently, reduce their sending rates. Fig 5 plots that cumulative distribution of the response times observed by the individual TCP connections within the Tmix aggregate. We find that in the presence of an aggressive high-speed transfer, the median response time of the Tmix transfers increases by 100 ms (and more for higher percentiles).
- The high-speed transfer over FAST achieves a fairly low throughput when it coexists with the Tmix cross traffic aggregate. The early response of FAST to delay-based congestion signals causes it to be too sensitive to the presence of loss-based TCP transfers; FAST is unable to adjust to, or effectively utilize, the dynamically-varying available bandwidth on the bottleneck link.
- RAPID is able to dynamically adapt to the changing available bandwidth left over by the low-speed TCP flows, while maintaining fairly low buffer occupancy. We conclude that RAPID is the only known high-speed protocol that can be deployed on regular Internet paths, without hurting the performance of conventional low-speed Internet transfers.[22]
  It is important to note again, however, that RAPID is unable to fully utilize the bandwidth which is available to it (as is evident by the "gap" between curves for the available bandwidth curve and RAPID throughput). As will be discussed in Section V-G, the utilization achieved by RAPID can be controlled by adjusting its parameters.

[21]Fig 4(1a) is plotted at an observation timescale of 50 ms. The actual traffic is significantly more bursty when observed at smaller timescales (1 ms or less), which are the timescales at which cross-traffic interacts with bandwidth-estimation within RAPID p-streams.



(a) (b)

Fig. 8. Impact of high-speed RAPID transfer on TMIX Traffic. (a) RAPID Throughput (with Tmix). (b) Tmix+RAPID Buffer.

F. Co-Existence With Loss-Based TCP Transfers

Our experiments above show that RAPID can efficiently co-exist with conventional Internet traffic—without adversely slowing down the low-speed transfers, while utilizing the dynamically-varying available bandwidth. Most of conventional traffic mixes contain short-lived transfers—it is also important to consider how RAPID works in the presence of competing *long-lived* transfers that might use other protocols.

It is worth noting that *no two high-speed congestion-control protocols* attain a fair share of bandwidth when they co-exist—this is true even when *both* protocols are loss-based [51], [52]. When one of the protocols is a delay-based protocol, prior studies have shown that due to their responsiveness to queue buildups, delay-based protocols typically achieve significantly

[22]The literature also contains congestion-control designs for large "background" transfers, that can also be used to transfer large data over regular Internet paths without hurting short-lived Internet transfers (e.g., [50]—to the best of our knowledge, however, none of these designs are targeted for high-speed environments, and are generally even more sluggish than regular TCP.

lower throughput than competing long transfers over loss-based protocols [53]–[56].[23] Note that, like most delay-based protocols, RAPID also responds to bottleneck queue build-ups much earlier and more frequently than loss-based protocols. We next conduct experiments to study its co-existence with long-lived transfers over loss-based protocols.

We ran a set of simple experiments in which a long RAPID transfer co-existed with a long transfer run on a different protocol—we experimented with each of the other high-speed protocols from the literature and found that RAPID attained only a significantly lower throughput. We conclude that RAPID is unable to compete against aggressive loss-based protocols that carry long-lived transfers. It is, however, important to note again that RAPID can be fairly efficiently co-deployed with conventional Internet traffic on shared Internet paths (as shown in Section V-E), which typically do not carry long-lived transfers over aggressive loss-based protocols.

G. Parameter Sensitivity

RAPID mechanisms are characterized by several parameters (Table I). In [45], we include results from an extensive set of controlled experiments that study the impact of these parameters on the performance of RAPID. In this section, we briefly discuss the parameters that are the most significant.

Our experiments reveal that the factor $\tau/(\eta * N)$ significantly influences the efficiency of RAPID in utilizing dynamically-varying end-to-end available bandwidth. A smaller value of τ allows RAPID transfers to quickly exploit the availability of additional bandwidth; while a larger $\eta * N$ makes RAPID more conservative about reacting to transient drop in available bandwidth.

In contrast, though, our detailed experimentation reveals that larger values of τ improve the fairness properties of RAPID, when multiple RAPID flows co-exist simultaneously. Thus, the parameter τ controls a tradeoff between the fairness and utilization properties of RAPID. In environments, where co-existence of multiple RAPID transfers is expected to be rare, a larger value of τ may be adopted to yield higher throughput.

Additionally, larger values of τ (and η) yield better stability of the aggregate traffic when multiple RAPID flows co-exist—this is to be expected, since τ and η control the low-pass filter used to smooth out available bandwidth estimates that may be highly variable.

Finally, the parameter m (spreadfactor) controls the range of rates probed for within a single p-stream. A larger value of m helps RAPID in quickly searching for avail-bw after a packet loss, or at flow initiation. However, a larger m also makes the p-stream structure more aggressive—the latter half of the p-stream is sent at much higher rates than the network can handle. On paths with limited bottleneck buffer space, such p-streams can significantly strain the queues and cause heavy packet losses, especially in dynamic traffic environments.

[23]Reference [56] demonstrates that it is possible to configure a given set of delay-based and loss-based protocol such that the former achieves higher throughput, but the configuration is highly sensitive to the operating environment.

VI. Open Issues

All of the performance gains reported in Section V have been observed with experiments conducted solely on the ns-2 simulator. There are at least three critical issues—the first two of which do not even exist in a simulator environment—that need further research to determine whether these gains are realizable in practice.

A. End-System Support

The paradigm requires the sender-side networking protocol stack to create *high-precision* and *fine-grained* inter-packet spacing. For instance, in order to probe for an avail-bw of 10 Gbps (and assuming a 1500 B path MTU), packets may have to be sent with an inter-packet spacing as small as a few microseconds—and this value reduces proportionally as we consider higher network speeds. High-precision is also needed in the receiver-side timestamping process that measures gaps between packets received. Inaccuracy in either of these processes can lead to incorrect conclusions about the end-to-end avail-bw. In a 10G network, for instance, an inaccuracy of the order of even $1\mu s$ could imply an rate-estimation inaccuracy of 50%! Unfortunately:

- Creating fine-grained and high-precision inter-packet gaps can prove to be fairly challenging in current software-based end-systems. Most operating systems (OSes) use an interrupt-driven model for managing and interleaving resources and computation. In this model, the process sending out packets of a p-stream may get interrupted at any arbitrary point of time while "waiting" for the required time-gap between two consecutive packets. Subsequently, the process may not regain control of the CPU before the transmission time of the next packet has elapsed. It is important to note that disabling interrupts while the process is waiting is a fairly inefficient option in servers that manage several processes simultaneously.
- Accurately timestamping packets when they arrive at the receiver is further complicated due to use of mechanisms such as *interrupt coalescence*, in which a NIC may wait for several packets to arrive before it interrupts the OS for processing these [57]—such mechanisms destroy packet gaps. Our evaluations in [58] show that such mechanisms can introduce delays of more than hundreds of μs! This is pretty disruptive for the fine-scale inter-packet gaps needed to accurately estimate ultra-high speed bandwidth.

Our Approach: In ongoing research, we are evaluating gap-creation using variable-sized dummy "gap" packets as in [59]—these packets are instantiated as Ethernet PAUSE frames, which are discarded by the first switch on the path that encounters them. By sizing such a frame appropriately, we can create the desired gap between two packets by inserting the frame in between, and by ensuring back-to-back transmission of all of the packets. Our evaluations suggest that this mechanism can help achieve inter-packet gap accuracy within $1\mu s$, even at 10 Gbps speeds.

For dealing with mechanisms that destroy gaps at small scales, we have developed a mechanisms, referred to as

Buffering-aware Spike Smoothing [58], that explicitly looks for the boundaries of buffering events that bunch packets up and destroy their inter-packet gaps—by averaging packet gaps strictly within the boundaries of individual buffering events, our mechanism is able to derive reliable bandwidth signatures from receive gaps (within 10% estimation error), without requiring the use of very large p-streams. We refer the reader to [58] for details.

B. Impact of Noise

The packet-scale paradigm fundamentally relies on the assumption that changes in inter-packet gaps of a small stream of packets can be used to robustly detect congestion. There are at least two phenomena that occur commonly in real systems and that can challenge this assumption:

- *Numerous occasions for non-bottleneck buffering in real systems:* Packets of a p-stream can be buffered at several points in their transit through even non-bottleneck resources. For instance, high-end switches often adopt some form of "burst-switching" mechanism, in which an incoming packet may not be switched immediately—the input NIC may wait and buffer a few closely-arriving packets before switching all of them to their corresponding outgoing NICs. Such buffering can destroy the gaps between packets of a p-stream, even before they reach the bottleneck router, R.
- *Transient queuing at the bottleneck link:* In a packet-switched network, traffic arrival can be fairly bursty at several timescales [60]–[62]. Even though the avail-bw may be large at longer timescales, the frequent arrival of short-timescale bursts can lead to low estimates of avail-bw by the p-streams these interact with. The packet-scale paradigm is especially vulnerable to this happening due to its ability to detect fine-scale changes in avail-bw.

Our Approach: In [58], we develop a noise-smoothing mechanism that explicitly detects boundaries of buffering events. Such events can be fairly diverse, and can include those that introduce significantly large queuing delays (such as interrupt coalescence as described before), as well as those that introduce smaller transient queues (such as cross-traffic burstiness)—consequently, we have found that detecting and smoothing out noise that occurs at such different timescales requires *multiple* passes of our mechanism, with each subsequent pass looking for buffering events that occur at larger timescales. Our evaluations in [58] show that our approach can help estimate bandwidth with less than 10% estimation error. More recently, we have also explored the suitability of machine learning approaches for bandwidth estimation that is robust to the presence of noise [63].

C. Environments With Small Bandwidth-Delay Products

The packet-scale paradigm is designed to alleviate the sluggishness of the RTT-scale framework in high speed environments. It is important to note that the larger the bandwidth-delay product of a network path, the larger would be the performance benefit offered by the paradigm. In particular, if data is transferred over a low speed path (with available bandwidth less than 100 Mbps), or over a short RTT path (as in data centers), there is little that the paradigm brings in terms of performance benefits. In fact, with the default configuration of probe-stream lengths in the RAPID prototype, on paths with small bandwidth-delay products, the probe-streams may well last longer than the path RTT![24] That would make RAPID even more sluggish than RTT-scale protocols.

Our Approach: In order to run on RAPID only on paths with high bandwidth-delay products, we propose to use two simple mechanisms. First, on initiation, all transfers rely on the default conventional window-based protocol configured on the sender operating system—once the congestion-window of the sender exceeds a threshold, th_{cwin}, the transfer switches to RAPID.[25] Second, when the bandwidth-estimation mechanism in RAPID consistenly returns bandwidth estimates lower than th_{AB}, RAPID is turned off and the window-based default protocol takes over—when congestion-window exceeds th_{cwin}, RAPID is turned back on. We are currently incorporating these mechanisms in a Linux-based implementation of RAPID. In our implementation, $th_{cwin} = 200Mbps * 100ms$, and $th_{AB} = 200Mbps * \frac{100ms}{r_{avg}}$.

D. Stability and Fairness in Heterogeneous and Dynamic Environments

The NS-2 experiments were conducted with only long-lived RAPID transfers and studied up to 100 simultaneous transfers sharing a bottleneck link. In practice, Internet links can aggregate tens-of-thousands (or more) of simultaneous transfers. Furthermore, these transfers can be fairly diverse in length, path RTTs, packet-arrival patterns, as well as number and types of links they traverse [64], [65]. It is crucial to understand how the packet-scale paradigm will interact with this heterogeneity and traffic dynamics. In particular, there are three critical issues that are somewhat unique to this paradigm (and may not impact RTT-scale protocols to the same extent):

- The packet-scale paradigm explicitly creates traffic burstiness at even sub-p-stream timescales (since packets within a p-stream are sent at fairly different rates). Further, the small timescales at which the paradigm is able to adapt the average rate of a transfer (once-per-p-stream) is likely to also impact the small-scale traffic dynamics of a mix of such transfers. For instance, Fig 3 shows that a mix of RAPID transfers creates a more bursty footprint of steady-state throughput than RTT-scale protocols. Before such a paradigm can be deployed world-wide, it is important to study and address any adverse impact this may have on Internet traffic dynamics and router queue-buildups.
- The packet-scale paradigm is able to adapt to changes in end-to-end avail-bw at fairly small timescales—this need not be all good. For instance, protocols that are very sensitive to load variations have sometimes also been found to be unstable and exhibit oscillatory behavior [66].

[24]For example, with $N = 90$, and packet sizes of 1000 B, a probe stream with $r_{avg} = 1Mbps$, would last 720 ms.

[25]Note that this mechanism is identical to what is adopted in *most* proposals for high-speed protocols.

While we did not observe any oscillations in any of our NS-2 experiments, this property can be guaranteed only through formal analysis.

- Would a pre-existing high-throughput transfer always yield to a newly arriving transfer (that initially samples low avail-bw and hence starts with a low rate)? We ask this question because while our NS-2 experiments yielded good fairness among RAPID transfers in many experiments (see Fig 3(b)), we also found some "aggressive" combinations of the protocol parameters with which simultaneous transfers would converge to an unequal allocation of the bottleneck bandwidth.

Our Approach: Due to the nature of the paradigm, properties such as stability, efficiency, and fairness can be comprehensively studied only by conducting fine-scale and closed-loop analysis of the interaction between p-streams as well shared buffers. While the literature is rich in closed-loop analyses of TCP-like protocols, such fine-scale analysis has not been attempted before. Our ongoing research suggests that the combination of simple small-scale models and experimental evaluation can help shed valuable insights into these properties [43].

VII. Discussion: Truly, a Paradigm Shift

Packet-scale congestion-control truly represents a paradigm shift from most prior work. It sheds the RTT-scale protocol operation; it sheds the ACK-based self-clocking that acts as a safety latch for other protocols; and it does not rely on estimation of buffer occupancy like most delay-based protocols. Instead, it adopts fine-scale packet timing for bandwidth probing as a first-order concept and holds the promise of being able to operate its control loop with close to optimally-minimal feedback delays (RTT). Needless to say, such a radically different design also has potential risks associated with it, which are being studied in our current research.

Related Protocols: It is important to clarify how some components of the paradigm are related to recent protocols. Some form of fine-scale probing has been adopted by Westwood [34], PCP [35], UDT [14], and NF-TCP [36]—each of these protocols, however, uses the idea to probe only every so often in an intermittent (non-continuous) manner. This can be fairly risky. For instance, when two transfers share a bottleneck link and their fine-scale probes do not overlap, each of them could conclude that a large amount of spare bandwidth is available [43]—each of them would end up sending huge volumes of traffic in the next RTT. As mentioned before, in ultra-high speed networks this could cause impairing network congestion. None of these protocols have been evaluated in a setting where multiple transfers share a multi-gigabit bottleneck link simultaneously—we believe these will simply not scale.

The idea of using self-congestion to estimate the end-to-end bandwidth has been developed in the bandwidth estimation literature [41], [42]—in fact, the protocols mentioned above also used this concept as one of their building blocks. In addition, delay-based protocols, such as Vegas [30], FAST [6], Illinois [10], and Compound [11] also use simpler versions of this concept. However, none of these protocols have successfully addressed the speed-overhead dilemma that fundamentally limits their scalability to ultra-high speed networks.

References

[1] S. Carter *et al. DOE Ultrascience Net: Experimental Ultra-Scale Network Testbed for Large-Scale Science*. [Online]. Available: http://www.csm.ornl.gov/ultranet/

[2] Network World. (Feb. 2010). *Facebook Sees Need for Terabit Ethernet*. [Online]. Available: http://www.networkworld.com/news/2010/020310-facebook-sees-need-for-terabit.html

[3] (Oct. 2010). *Tomorrow's Internet: 1000 Times Faster*. [Online]. Available: http://engineering.ucsb.edu/news/468/

[4] Verizon Wireless. (Mar. 2011). *Verizon First to Deploy Standards-Based 100G Ethernet on Long-Haul IP Backbone Network*. [Online]. Available: http://newscenter.verizon.com/press-releases/verizon/2011/verizon-first-to-deploy.html

[5] *Google Fiber*. http://www.google.com/fiber/kansascity/index.html

[6] D. Wei, C. Jin, S. Low, and S. Hegde, "FAST TCP: Motivation, architecture, algorithms, performance," *IEEE/ACM Trans. Netw.*, vol. 14, no. 6, pp. 1246–1259, Dec. 2006.

[7] S. Floyd, "Highspeed TCP and quick-start for fast long-distance networks," in *Proc. Plenary Talk 1st Int. Workshop Protocols Fast Long-Distance Netw.*, Feb. 2003, pp. 1–29.

[8] T. Kelly, "Scalable TCP: Improving performance in highspeed wide area networks," in *Proc. 1st Int. Workshop Protocols Fast Long-Distance Netw.*, Feb. 2003, pp. 1–8.

[9] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control (BIC) for fast long-distance networks," in *Proc. IEEE INFOCOM*, Mar. 2004, pp. 2514–2524.

[10] S. Liu, T. Başar, and R. Srikant, "TCP-Illinois: A loss- and delay-based congestion control algorithm for high-speed networks," *Perform. Eval.*, vol. 65, nos. 6–7, pp. 417–440, Jun. 2008.

[11] K. Tan, J. Song, Q. Zhang, and M. Sridharan, "A compound TCP approach for high-speed and long distance networks," in *Proc. IEEE INFOCOM*, Apr. 2006, pp. 1–12.

[12] S. Ha, I. Rhee, and L. Xu, "CUBIC: A new TCP-friendly high-speed TCP variant," *ACM SIGOPS Oper. Syst. Rev.*, vol. 42, no. 5, pp. 64–74, Jul. 2008.

[13] D. Leith and R. Shorten, "H-TCP: TCP for high-speed and long-distance networks," in *Proc. PFLDnet*, 2004, pp. 1–16.

[14] Y. Gu and R. L. Grossman, "UDT: A high performance data transport protocol," Ph.D. dissertation, Dept. Comput. Sci., Univ. Illinois Chicago, Chicago, IL, USA, 2005.

[15] Caltech Media Relations. (Nov. 2007). *High Energy Physicists Set New Record for Network Data Transfer*. [Online]. Available: http://media.caltech.edu/press_releases/13073

[16] N. S. V. Rao, W. Yu, W. R. Wing, S. W. Poole, and J. S. Vetter, "Wide-area performance profiling of 10GigE and InfiniBand technologies," in *Proc. Int. Conf. High Perform. Comput., Netw., Storage Anal.*, Nov. 2008, pp. 1–12.

[17] W. Allcock *et al.*, "Grid-enabled particle physics event analysis: Experiences using a 10 Gb, high-latency network for a high-energy physics application," *Future Generat. Comput. Syst.*, vol. 19, no. 6, pp. 983–997, 2003.

[18] C. Jin *et al.*, "FAST TCP: From theory to experiments," *IEEE Netw.*, vol. 19, no. 1, pp. 4–11, Jan./Feb. 2005.

[19] V. Konda and J. Kaur, "RAPID: Shrinking the congestion-control timescale," in *Proc. IEEE INFOCOM*, Apr. 2009, pp. 1–9.

[20] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, document IETF-RFC3550, 2003.

[21] H. Schulzrinne, A. Rao, R. Lanphier, M. Westerlund, and M. Stiemerling, *Real Time Streaming Protocol 2.0 (RTSP)*, document IETF-RFC2326, 2014.

[22] *Packet-Scale Congestion Control*. [Online]. Available: http://rapid.web.unc.edu

[23] *Network Simulator-2 ns-2*. [Online]. Available: http://www.isi.edu/nsnam/ns/

[24] D. Katabi, M. Handley, and C. Rohrs, "Congestion control for high bandwidth-delay product networks," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, Pittsburgh, PA, USA, Aug. 2002, pp. 89–102.

[25] R. Kumar and J. Kaur, "Towards a queue sensitilve transport protocol," in *Proc. IEEE IPCCC*, Dec. 2008, pp. 319–326.

[26] N. Dukkipati, "Rate control protocol (RCP): Congestion control to make flows complete quickly," Ph.D. dissertation, Dept. Elect. Eng., Stanford Univ., Stanford, CA, USA, Oct. 2007.
[27] Y. Xia, L. Subramanian, I. Stoica, and S. Kalyanaraman, "One more bit is enough," in *Proc. ACM SIGCOMM*, Philadelphia, PA, USA, Aug. 2005, pp. 37–48.
[28] A. Kuzmanovic, "The power of explicit congestion notification," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, Philadelphia, PA, USA, Aug. 2005, pp. 61–72.
[29] K. Fall and S. Floyd, "Simulation-based comparisons of Tahoe, Reno and SACK TCP," *ACM Comput. Commun. Rev.*, vol. 26, no. 3, pp. 5–21, Jul. 1996.
[30] L. S. Brakmo, S. W. O'Malley, and L. L. Peterson, "TCP Vegas: New techniques for congestion detection and avoidance," in *Proc. ACM SIGCOMM*, Aug. 1994, pp. 24–35.
[31] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," in *Proc. ACM SIGCOMM*, Aug. 2000, pp. 43–56.
[32] F. P. Kelly, A. K. Malullo, and D. K. H. Tan, "Rate control in communication networks: Shadow prices, proportional fairness and stability," *J. Oper. Res. Soc.*, vol. 49, pp. 237–252, Mar. 1998.
[33] S. Kunniyur and R. Srikant, "End-to-end congestion control schemes: Utility functions, random losses and ECN marks," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 1323–1332.
[34] S. Mascolo, C. Casetti, M. Gerla, M. Y. Sanadidi, and R. Wang, "TCP westwood: Bandwidth estimation for enhanced transport over wireless links," in *Proc. ACM MobiCom*, 2001, pp. 287–297.
[35] T. Anderson, A. Collins, A. Krishnamurthy, and J. Zahorjan, "PCP: Efficient endpoint congestion control," in *Proc. 3rd Symp. Netw. Syst. Design Implement. (NSDI)*, May 2006, pp. 1–14.
[36] M. Arumaithurai, X. Fu, and K. K. Ramakrishnany, "NF-TCP: Network friendly TCP," in *Proc. IEEE Workshop Local Metropolitan Area Netw.*, May 2010, pp. 1–6.
[37] E. Gavaletz and J. Kaur, "Decomposing RTT-unfairness in transport protocols," in *Proc. IEEE Workshop Local Metropolitan Area Netw.*, May 2010, pp. 1–6.
[38] S. Floyd, *Congestion Control Principles*, document RFC 2914, Sep. 2000. [Online]. Available: http://www.ietf.org/rfc/rfc2914.txt
[39] F. Hernández-Campos, F. Donelson, and S. K. Jeffay, "Generating realistic TCP workloads," in *Proc. CMG*, Dec. 2004, pp. 273–284.
[40] Aberdeen Group. (Nov. 2011). *Why Web Performance Matters: Is Your Site Driving Customers Away?* [Online]. Available: http://www.mcrinc.com/Documents/Newsletters/201110_why_web_performance_matters.pdf
[41] M. Jain and C. Dovrolis, "End-to-end available bandwidth: Measurement methodology, dynamics, and relation with TCP throughput," *IEEE/ACM Trans. Netw.*, vol. 11, no. 4, pp. 537–549, Aug. 2003.
[42] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathChirp: Efficient available bandwidth estimation for network paths," in *Proc. Passive Active Meas. Workshop*, Apr. 2003, pp. 1–11.
[43] E. Gavaletz and J. Kaur, "What happens when many probe for bandwidth simultaneously?" Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, Tech. Rep., Nov. 2010.
[44] P. Haga, A. Pasztor, D. Veitch, and I. Csabai, "Pathsensor: Towards efficient available bandwidth estimation," in *Proc. 3rd Int. Workshop Internet Perform., Simulation, Monitor., Meas.*, Mar. 2005, pp. 39–48.
[45] R. Lovewell, Q. Yin, T. Zhang, J. Kaur, and F. Smith, "The packet-scale congestion control paradigm," Dept. Comput. Sci., Univ. North Carolina Chapel Hill, Chapel Hill, NC, USA, Tech. Rep., Jul. 2014.
[46] S. Gorinsky and H. Vin, "Additive increase appears inferior," Dept. Comput. Sci., Univ. Texas Austin, Austin, TX, USA, Tech. Rep., 2000.
[47] M. Vojnovic, J.-Y. Le Boudec, and C. Boutremans, "Global fairness of additive-increase and multiplicative-decrease with heterogeneous round-trip times," in *Proc. IEEE INFOCOM*, Mar. 2000, pp. 1303–1312.
[48] T. Cui and L. Andrew. *FAST TCP Simulator Module for ns-2, Version 1.1*. [Online]. Available: http://www.cubinlab.ee.unimelb.edu.au/ns2fasttcp
[49] M. C. Weigle, P. Adurthi, F. Hernández-Campos, K. Jeffay, and F. D. Smith, "Tmix: A tool for generating realistic TCP application workloads in ns-2," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 3, pp. 65–76, Jul. 2006.
[50] A. Venkataramani, R. Kokku, and M. Dahlin, "TCP Nice: A mechanism for background transfers," in *Proc. 5th Symp. Oper. Syst. Design Implement. (OSDI)*, Boston, MA, USA, Dec. 2002, pp. 329–343.
[51] S. Molnár, B. Sonkoly, and T. A. Trinh, "A comprehensive TCP fairness analysis in high speed networks," *Comput. Commun.*, vol. 32, nos. 13–14, pp. 1460–1484, Aug. 2009.
[52] D. Miras, M. Bateman, and S. Bhatti, "Fairness of high-speed TCP stacks," in *Proc. IEEE 22nd Int. Conf. Adv. Inf. Netw. Appl.*, Mar. 2008, pp. 84–92.
[53] A. Hayes David and G. Armitage, "Improved coexistence and loss tolerance for delay based TCP congestion control," in *Proc. 35th Annu. IEEE Conf. Local Comput. Netw.*, Oct. 2010, pp. 24–31.
[54] L. Budzisz, R. Stanojevic, R. Shorten, and F. Baker, "A strategy for fair coexistence of loss and delay-based congestion control algorithms," *IEEE Commun. Lett.*, vol. 13, no. 7, pp. 555–557, Jul. 2009.
[55] Ł. Budzisz, R. Stanojevic, A. Schlote, F. Baker, and R. Shorten, "On the fair coexistence of loss- and delay-based TCP," *IEEE/ACM Trans. Netw.*, vol. 19, no. 6, pp. 1811–1824, Dec. 2011.
[56] A. Tang, J. Wang, S. Hegde, and S. H. Low, "Equilibrium and fairness of networks shared by TCP Reno and Vegas/FAST," *Telecommun. Syst.*, vol. 30, no. 4, pp. 417–439, Dec. 2005.
[57] R. Prasad, M. Jain, and C. Dovrolis, "Effects of interrupt coalescence on network measurements," in *Proc. 5th Int. Workshop PAM*, 2004, pp. 247–256.
[58] Q. Yin, J. Kaur, and F. D. Smith, "Can bandwidth estimation tackle noise at ultra-high speeds?" in *Proc. 22nd IEEE Int. Conf. Netw. Protocols*, Raleigh, NC, USA, Oct. 2014, pp. 107–118.
[59] R. Takano *et al.*, "Design and evaluation of precise software pacing mechanisms for fast long-distance networks," in *Proc. PFLDnet*, 2005, pp. 1–7.
[60] A. Feldmann, A. C. Gilbert, P. Huang, and W. Willinger, "Dynamics of IP traffic: A study of the role of variability and the impact of control," in *Proc. ACM SIGCOMM*, 1999, pp. 301–313.
[61] H. Jiang and C. Dovrolis, "Source-level IP packet bursts: Causes and effects," in *Proc. 3rd ACM SIGCOMM Conf. Internet Meas. (IMC)*, Oct. 2003, pp. 301–306.
[62] Z.-L. Zhang, V. J. Ribeiro, S. Moon, and C. Diot, "Small-time scaling behaviors of Internet backbone traffic: An empirical study," in *Proc. IEEE INFOCOM*, Apr. 2003, pp. 1826–1836.
[63] Q. Yin and J. Kaur, "Can machine learning benefit bandwidth estimation at ultra-high speeds?" in *Proc. Passive Active Meas. Workshop (PAM)*, Apr. 2016, pp. 397–411.
[64] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay, "Variability in TCP round-trip times," in *Proc. ACM SIGCOMM Internet Meas. Conf.*, Oct. 2003, pp. 279–284.
[65] F. D. Smith, F. Hernández-Campos, K. Jeffay, and D. Ott, "What TCP/IP protocol headers can tell us about the Web," in *Proc. ACM SIGMETRICS*, Jun. 2001, pp. 245–256.
[66] Z. Wang and J. Crowcroft, "Analysis of shortest-path routing algorithms in a dynamic network environment," *ACM Comput. Commun. Rev.*, vol. 22, no. 2, pp. 63–71, Apr. 1992.

Rebecca Lovewell, photograph and biography not available at the time of publication.

Qianwen Yin, photograph and biography not available at the time of publication.

Tianrong Zhang, photograph and biography not available at the time of publication.

Jasleen Kaur, photograph and biography not available at the time of publication.

Frank Donelson Smith, photograph and biography not available at the time of publication.