Can Bandwidth Estimation Tackle Noise at Ultra-High Speeds?

Qianwen Yin Jasleen Kaur F. Donelson Smith
Department of Computer Science
University of North Carolina at Chapel Hill

***Abstract*—While existing bandwidth estimation tools have been shown to perform well on 100Mbps networks, they fail to do so at gigabit and higher network speeds. This is because finer inter-packet gaps are needed to probe for higher rates—fine gaps are more susceptible to be disturbed by small-scale buffering-related noise. In this paper, we evaluate existing noise-reduction techniques for tackling the issue, and show that they are ineffective on 10Gbps links. We propose a novel smoothing strategy, *Buffering-aware Spike Smoothing (BASS)*, which can be applied effectively to both single-rate and multi-rate probing frameworks and help significantly in scaling bandwidth estimation to ultra-high speed networks. Besides, we provide first evidence that accurate bandwidth estimation using our strategy can help improve the performance of congestion-control protocols on real 10Gbps networks.**

I. Introduction

Two trends motivate the need for bandwidth estimation at ultra-high speeds:[1].

- *Why estimate bandwidth?* End-to-end available bandwidth has been found to be an important metric in several application domains–including server selection [1], [2], [3], overlay routing [4], [5], TCP configuration [6], media-streaming protocols [7], [8], and more recently, high-speed congestion-control [9], [10], [11]. Consequently, the last decade has witnessed a rapid growth in the design of bandwidth estimation techniques [12], [13], [14], [15], [16].
- *Why consider ultra-high speeds?* Boosted by the explosion of multimedia services and latest communication technology, Google fiber project is bringing Gigabit access links to homes [17]. The next generation of network speed is around the corner, urging bandwidth estimation techniques to scale to ultra-high speeds.

Unfortunately, although existing bandwidth estimation tools have been shown to perform well on 100Mbps networks, they fail to do so at gigabit and higher network speeds [18]. This is because small inter-packet gaps are needed for probing higher bandwidth—such fine-scale gaps are more susceptible to being disturbed by small-scale buffering related noise at shared resources and at end hosts [19].

This material is based upon work supported by the National Science Foundation under Awards CNS-1018596 and OCI-1127413

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

978-1-4799-6204-4/14$31.00

Several mechanisms have been proposed for reducing the impact of noise in [20], [21], [22], but these have been evaluated only on 100Mbps networks. [23] developed a hardware assisted system to achieve accurate bandwidth estimation on 10Gbps links, but no software solutions exist so far. In this paper, we evaluate the state of the art and develop novel strategies to address noise on ultra-high speed networks. The main contributions of our work are:

- We evaluate existing bandwidth estimation strategies at ultra high speeds, using a 10Gbps testbed. We find that the best-performing strategies require a large number of probing packets in order to scale up to 10 Gbps links.
- We propose *Buffering-aware Spike Smoothing (BASS)*, a smoothing mechanism that helps achieve robustness to noise with low probing overhead, even on 10 Gbps links. To the best of our knowledge, we are the *first* to show that bandwidth estimation can effectively scale to ultra high-speed links, free of any hardware assistance.
- Last but not least, we provide the first evidence of the efficacy of bandwidth estimation in the context of an ultra-high speed congestion-control protocol. Bandwidth estimation-based protocols have been readily shunned due to the distrust of avail-bandwith or delay measurements in the presence of noise. However, we find that, armed with the ability to accurately estimate and track avail-bw, such congestion-control protocols can outperform their loss-based counterparts.

In the rest of this paper, we summarize background on bandwidth estimation in Section II and our experimental setup in Section III. Section IV discusses and evaluates recent noise-reduction strategies. In Section V we present our smoothing strategy. In Section VI we develop multi-pass spike removal for multi-rate probing frameworks. In Section VII we incorporate and evaluate our strategy in an bandwidth estimation-based congestion-control protocol. Related work is summarized in Section VIII and our conclusions in Section IX.

II. Background: PRM-based Bandwidth Estimation

The past decade has witnessed a rapid growth in the design of techniques for estimating available bandwidth [15], [13], [24], [14]. Existing bandwidth estimation techniques base their bandwidth estimation logic on two prominent models, namely, the probe gap model and the probe rate model (PRM) [25]. Tool evaluations have shown that PRM tools are more robust in the presence of multiple congested links [26], [18]—in this

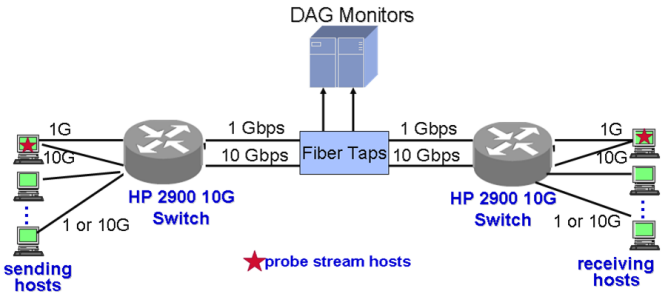


Fig. 1: Laboratory Testbed Configuration

paper, we focus on these. Below, we briefly summarize the PRM approach.

PRM tools typically send multiple packets at the same probing rate, in groups commonly referred to as *probe-streams*—probing rate, r_i, of the ith packet is achieved by controlling the inter-packet gap as: $g_i^s = \frac{p_i}{r_i}$, where p_i is the size of the ith packet and g_i^s is the sendgap between packet $i-1$ and i. We use N to denote the length of a probe-stream, in terms of number of packets.

PRM tools rely on the principle of *self-congestion*, according to which: if $r_i > AB$, then $q_i > q_{i-1}$, where q_i is the queuing delay experienced by the ith packet at the bottleneck link, and AB is the available bandwidth on that link. Existing tools try out different probing rates and search for the highest rate r_{max} that does *not* result in increasing queueing delays. Assuming fixed routes and constant processing delays, increasing trends in queuing delays can be detected either by finding increasing trends in the end-to-end one-way delays [27], or by observing if the inter-packet receive gaps (or packet arrival rate at receiver) are higher than the send gaps (or packet sending rate) [28], [21].

Existing tools differ in the strategy used for searching r_{max}. Many tools rely on iterative feedback-based binary search, in which senders wait for receiver feedback on a probe-stream that has been sent, and either halve or double the probing rate for the next probe-stream, depending on whether or not self-congestion is detected—Pathload is the most prominent of such tools [15]. Some tools rely on multi-rate non-feedback based probing, in which probe-streams spanning several probing rates are sent back-to-back and bandwidth estimated based on the highest probing rate that did not result in self-congestion [29], [13].

In this paper, we will evaluate PRM strategies, based on how accurately they can infer the occurence of self-congestion.

III. Experimental Setup

A salient feature of our analysis methodology in this paper is that all of the evaluations (of both state of the art techniques as well as our proposals) are performed on a 10 Gbps testbed. We begin by first describing our experimental setup—for improved readability, some details are included only in Appendix A.

a) Testbed: For our experiments, we use the dedicated network illustrated in Fig 1. The switch-to-switch path is a 10 Gbps fiber path. The two end hosts involved in bandwidth estimation are connected to either sides of the switches using 10 Gbps Ethernet. The network includes an additional 10 pairs of hosts (sender and receiver) that are used to generate cross traffic sharing the switch-to-switch link.

b) Recording Available Bandwidth: To measure the ground-truth about avail-bw, we collect packet traces on the fiber links between the two switches using fiber splitters attached to Endace DAG monitoring NICs—these NICs provide timestamps with nanosecond precision and 10 nanosecond accuracy. A complete trace of each experiment was obtained from the DAG monitor between the switches. This trace was used to count the number of cross-traffic bits encountered in any given time interval—subtracting this from the bottleneck link capacity yielded the *ground truth available bandwidth* at the bottleneck link in that interval.

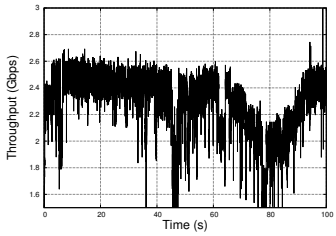


Fig. 2: B-CT: Bursty Cross Traffic

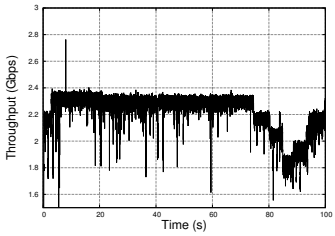
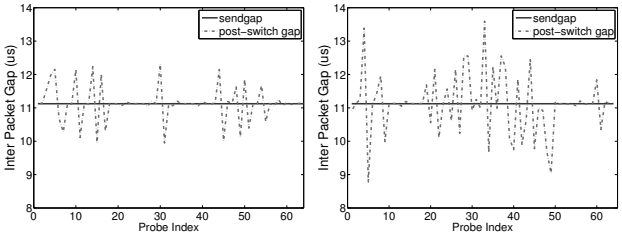


Fig. 3: S-CT: Smooth Cross Traffic

c) Cross Traffic Generation: In order to generate representative bursty traffic loads on the bottleneck link between the two switches, we used 10 pairs of sending and receiving hosts running a locally-modified version of the SURGE program for generating synthetic web traffic [30]. An important consideration for comparisons among different methods for bandwidth estimation is that the cross traffic be *consistently* controlled across all experiments. In particular, the cross traffic should not be responsive to the amount of bandwidth used by the probe packets. The SURGE program uses TCP connections with an MTU size of 1500 bytes for emulating web browsers and servers. To eliminate the responsive behavior of TCP, we recorded a packet trace of the SURGE data source (emulated server) to be used as input to the *tcpreplay* program [31] during experiments.[2] The aggregate traffic for the 10 pairs of SURGE hosts recorded by the DAG between the switches is shown in Fig 2, plotted in 10 ms intervals. Even at this relatively large time scale, the traffic clearly has a highly variable and bursty character—we refer to this traffic as B-CT.

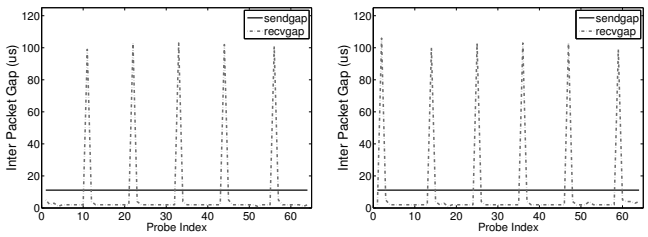
Some of our experiments were also conducted with a smoothed version of the tcpreplay traffic. The smoothing was implemented by running the token bucket Qdisc included in Linux on each host pair. The maximum burst rate was limited to 5 percent of the average sending rate for that pair along with a buffer large enough to prevent any loss at the token bucket. The resulting aggregate traffic for the 10 pairs recorded by the

[2]Specifically, each of the 10 pairs of SURGE hosts was run for at least 10 minutes and a packet trace obtained for each pair. During an experiment, each pair of hosts then ran tcpreplay to generate traffic from the trace recorded for that pair.



(a) with CBR (b) with B-CT

Fig. 4: Gaps Observed After the Switch



(a) with CBR (b) with B-CT

Fig. 5: Gaps Observed at Receiver

DAG between the switches is shown in Fig 3, plotted in 10 ms intervals. Clearly this smoothing eliminated most of the peaks in bandwidth utilization by the cross traffic—we refer to this traffic as S-CT. Both B-CT and S-CT have an average load of around 2.4 Gbps.

d) Generating (and Receiving) Probe Streams: In our experiments, the *iperf* client program is used as a source of data segments with an MTU size of 9000 bytes. To turn the stream of segments sourced by iperf into probe streams, we wrote a Linux Qdisc packet scheduler that sits between the bottom of IP and the NIC device driver and creates inter-packet gaps within probe-streams. This Qdisc created probe streams of a given size (e.g. 64 packets) and probing rate (e.g. 6 Gbps) by computing the proper inter-packet gap to achieve the desired average rate for the stream.

We rely on Ethernet PAUSE frames to enforce inter-packet send-gaps (details are in Appendix A)—our measurements show that 90% of the created gaps were within 1μs of the intended gaps. At the receiver, we timestamp packets with microsecond precision in an ingress Qdisc (logically positioned in the Linux networking stack between the device driver and the bottom of IP).

In the experiments, the average send rate of individual probe streams starts at 5.0 Gbps and increases in 0.5 Gbps increments up to 9 Gbps. Each probe stream is separated from the previous by a 10 millisecond idle time with no packets sent to eliminate any correlation in queueing time between streams. This sequence of probe streams is repeated to obtain *thousands* of observations for each of the probing rates.

e) Computing Metrics: To compute bandwidth estimates, we used an off-line program that implements the various algorithm for computing bandwidth estimates studied in this paper. The inputs to this algorithm were the logs from the kernel modules that recorded sequence numbers, intended send gaps, and receive gaps. Only probe streams that were complete (had the correct length and did not experience packet drops) were used as input.

IV. State of the Art

A. Issues in High Speed Networks: Noise!

Bandwidth estimation has been shown to work well on paper and in simulation settings, but independent real-world evaluations have observed poor performance in practice [18]. One key factor in real networks is the presence of small-scale buffering at the several instances of shared system resources between the sender and the receiver—such buffering introduces *noise* in the per-packet delays, and can corrupt the analysis of one-way delays or receiver-side inter-packet gaps for estimating available bandwidth. Such noise has an even higher impact in ultra high speed networks, in which the delays and gaps are much *finer* in scale.

There are at least two types of noise sources encountered:

- *Burstiness in competing-traffic at bottleneck resources:* If the available bandwidth at bottleneck resources varies at short-timescales due to burstiness in competing traffic, then all packets within a probe-stream may not consistently show an increasing trend in queueing delays. Fig 4 plots, for a sample probe-stream, the constant send-gaps, as well as the gaps observed by the DAG monitors on the shared bottleneck link with (i) constant bit-rate (CBR) cross traffic, and with (ii) bursty replayed traffic, B-CT. We find that bursty cross traffic can distort the inter-packet gaps in a probe-stream in a fairly noisy manner—such distortions can make it challenging to make a reliable conclusion about the available bandwidth.
- *Transient queuing at non-bottleneck resources:* Even though a resource may not be a bottleneck on the path of a packet, it can certainly induce short-scale transient queues when it is temporarily unavailable while servicing competing processes or traffic. In our testbed topology of Fig 1, this can happen, for instance, while accessing the high-speed cross-connects at the switches, or while waiting for CPU processing after packets arrive at the receiver-side NIC. In fact, *interrupt coalesence*, which is turned on by default in receivers, can force packets to wait even if the CPU is available [32]—the waiting time can be significant compared to the fine-scale gaps needed in ultra high-speed networks.

  Fig 5 plots, for the same probe-streams, the send-gaps, as well as the gaps observed as soon as the pacekts are received by the OS software (running on the CPU) at the receiver in our topology. We find that interrupt-coalescence can significantly further distort the inter-packet gaps observed previously in Fig 4. Furthermore, this can happen both for smooth as well as bursty cross-traffic—that is, the influence of interrupt coalescence

significantly outweighs that of cross-traffic burstiness.

B. State of the Art: Dealing with Noise

Noise has been previously recognized as a potential issue for bandwidth estimation in practice. Several smoothing strategies have been proposed to deal with it. Pathload [33] attempts to identify those probes that arrive within a single interrupt burst[3]—it then eliminates all coalesced probes from the burst, except the last one that exeriences the minimum queuing delay at the receiver NIC.

However, [20] observed that Pathload is unable to estimate bandwidth correctly in the presence of non-negligible interrupt delays ($> 125\mu s$) in 100 Mbps networks. To address this, it introduced IMR-Pathload, that first applied signal denoising techniques to the one-way delays (OWDs) observed by Pathload. Two different techniques were proposed and evaluated: multi-level discrete wavelet transform, and fixed-window ($\frac{1}{10}$th of probe stream length) worth of averaging of consecutive one-way delays—in this paper, we refer to these techniques as *IMR-wavelet* and *IMR-avg*, respectively. Both techniques were reported to significantly improve upon the trend-detection accuracy of Pathload with noisy OWDs.

PRC-MT [21] used the idea of sending (and averaging over) longer probe-streams to reduce the impact of noise in real-world probe-streams. This tool estimates available bandwidth by finding the largest probing rate (r_{in}) such that the corresponding packet arrival rate (r_{out}) at the receiver is not lower than it. To deal with noise, the tool first tunes for the probe-stream length, N—it does so by sending packets at the Asymptotic Dispersion Rate (which is proven to be larger than the available bandwidth), and searching for the smallest N such that increasing N further does not impact the average receive rate [21].

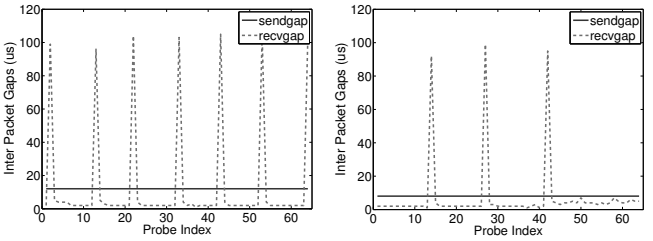
C. How Well Do These Work?

A fundamental building block used by all PRM tools is the decision of: *whether or not a given probing rate is larger than the available bandwidth.* We evaluate the state of the art techniques described above for their ability to make this decision correctly in the presence of noise. For this, we run testbed experiments in the presence of the B-CT cross traffic, and emulate *several thousands* of probe-streams of different lengths (N = 32, 64, 128, 320 packets) and different probing rates (5 - 9 Gbps).[4] We then collect inter-packet gaps right after the shared switch link (using the DAG monitor) as well as those observed at the receiver. The DAG also helps us collect ground-truth about the available bandwidth encountered by *each* probe-stream.

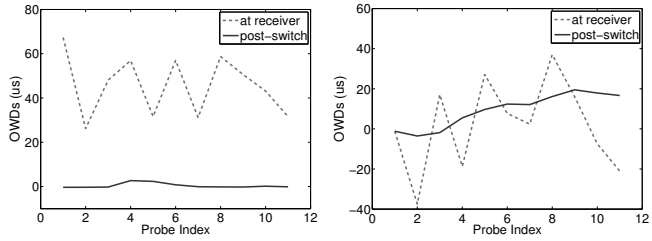
We apply the noise-reduction techniques described above to each probe-stream and evaluate their accuracy in estimating whether or not the corresponding probing rate was higher than the available bandwidth. Fig 6 summarizes the inaccuracy

[3]Our testbed measurements (e.g., Fig 5) show that *all* probes, in fact, encounter an interrupt burst.

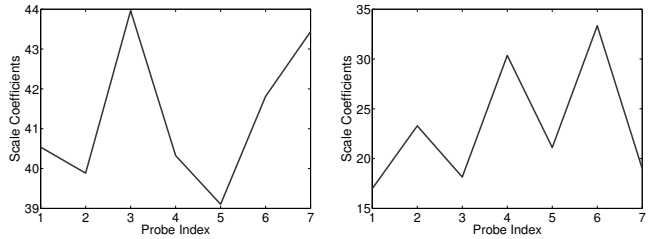
[4]We first run the N-selection logic of PRC-MT on our testbed, which dictates that probe-streams should be of length 320 packets.



(a) Probing at 6Gbps, AB: 7.5Gbps (b) Probing at 9Gbps, AB: 8.4Gbps



(c) Probing at 6Gbps, IMR-avg (d) Probing at 9Gbps, IMR-avg



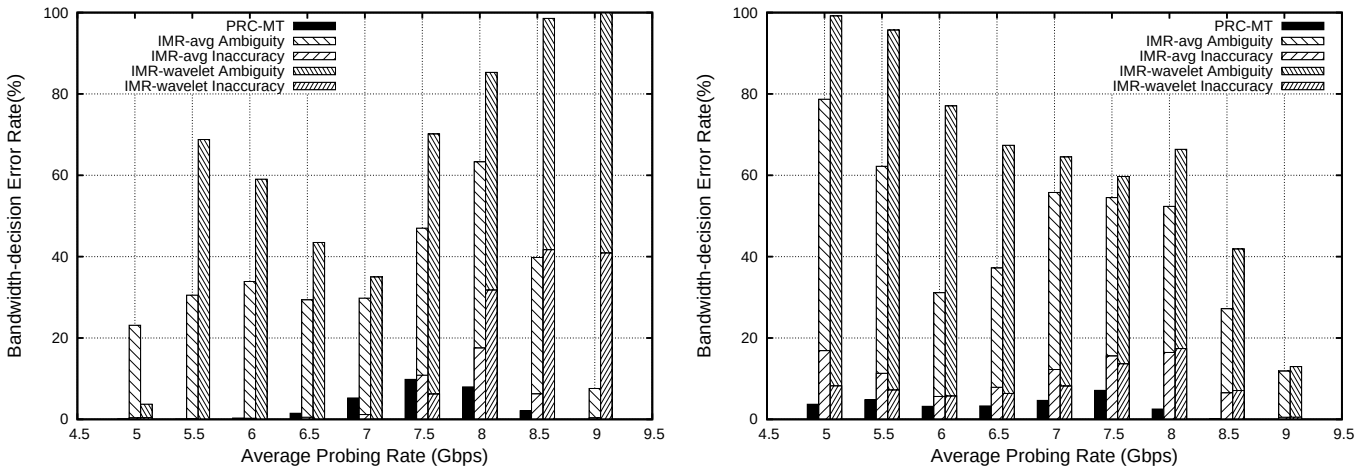
(e) Probing at 6Gbps, IMR-wavelet (f) Probing at 9Gbps, IMR-wavelet

Fig. 7: Two example probe streams smoothed by IMR-Pathload.

results for probe-streams with different average probing rates, different smoothing strategies, and the probe-stream length of 320.[5] We compute the inaccuracy using both, gaps observed at the receiver as well as those observed after crossing the bottleneck switch (which do not include impact of receiver-side interrupt coalescence). We find that:

- PRC-MT is fairly accurate when each probe-stream consists of 320 packets. More than 90% of probe-streams accurately infer whether the probing rate is higher or lower than the available bandwidth—this is despite the presence of bursty cross-traffic. PRM-MT performs well whether used with receiver-side gaps, or with gaps observed immediately after the bottleneck switch.
- The IMR techniques fail to arrive at a decision for a huge fraction of probe-streams (see the *ambiguity* bars in Fig 6). For the probe-streams for which they do provide an answer, IMR-avg has much lower accuracy than PRC-MT, even when probe-streams are as long as 320 packets. For the wavelet we picked (Daubechies's length 4 wavelet up to level 3 as recommended in [21]), IMR-wavelet

[5]The *bandwidth-decision error rate* plotted in Fig 6 represents the fraction of probe-streams that yielded incorrect decisions to the question—whether or not a given probing rate is larger than the available bandwidth. For IMR-avg and IMR-wavelet, we also include bars for the fraction of probe-streams that were unable to arrive at a decision (ambiguous).



(a) Using receiver-side gaps

(b) Using post-switch gaps

Fig. 6: Bandwidth-decision Errors: State of the Art

yields inaccurate results in all experiments (and like IMR-avg, fails to arrive at any decision for a large fraction of probe-streams).

Furthermore, Fig 6 shows that IMR-avg and IMR-wavelet do not work well even without interrupt coalescence (when post-switch gaps are used).

For two example probe streams, we collect one-way delays observed at the receiver, apply IMR-avg or IMR-wavelet on them, and plot the smoothed OWDs in Fig 7(c)-(f)—we find that despite the smoothing, these retain a saw-tooth pattern due to the impact of heavily coalesced arrivals. We believe this is primarily due to the use of the PCT and PDT metrics, which are not robust in ultra-high speed networks, despite smoothing using the IMR-pathload techniques. In the rest of this paper, we focus only on PRC-MT, which significantly outperforms the others *given sufficiently long probe streams.*[6]

We next evaluate PRC-MT when shorter probe streams are used. Fig 8 plots the inaccuracy results for $N = 32, 64, 128, 320$ packets. We find that PRC-MT yields fairly inaccurate results for probe-streams that are shorter than 320 packets. Probe streams of length 320 are pretty large for many applications, especially those that need to probe regularly—consider for example, the application domain of ultra-high speed congestion control. We next ask: *can light-weight/quick probing be made to work at ultra-high speeds?*

V. Buffering-aware Spike Smoothing

PRC-MT works well in reducing the impact of noise when probe streams are fairly large, but fails to work when probe-streams are small—this suggests that there is some underlying signature in the one-way delays that can be recovered robustly only in large probe-streams. It is natural to wonder: why can such signatures not be recovered when probe-streams are small?

[6]In fact, we have conducted all subsequent evaluations even with IMR-ag and IMR-wavelet—while the smoothing mechanism we propose in Section V helps improve their performance, these still perform significantly worse than PRM-MT.

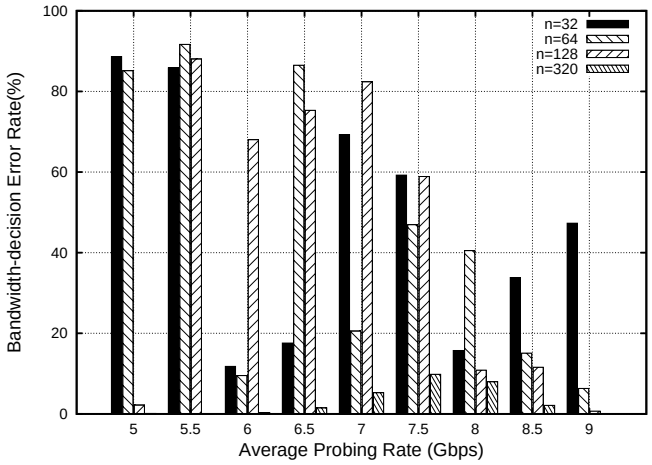


Fig. 8: PRC-MT: Impact of N on Bandwidth Estimation Error

To understand this, we define the concept of a *buffering event*. Consider what the "spikes" and "dips" in Fig 7(a)-(b) represent. When a system resource (such as transmission on the bottleneck link or processing by the receiver-side CPU) becomes temporarily unavailable due to competing traffic or processes, packets queue up in a system queue waiting for access to the resource. This causes a large gap ("spike") for the first packet in this queue. When the resource does become available, the remaining packets in the queue get processed fairly quickly (especially for non-bottleneck resources), and

Algorithm 1 Buffering Aware Spike Smoothing

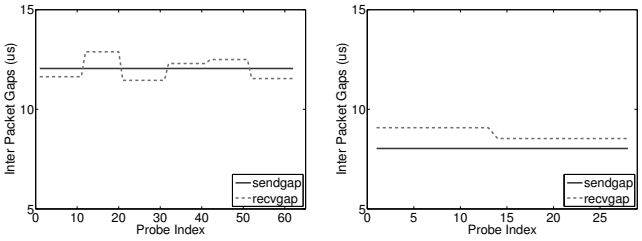
```
1: function AVERAGEPROBESTREAM(spike_begin, spike_end)
2:     sum_sendgap ← 0, sum_recvgap ← 0
3:     if spike_begin < spike_end then
4:         for i = spike_begin → spike_end do
5:             sum_sendgap+ = send_gap[i]
6:             sum_recvgap+ = recv_gap[i]
7:         for i = spike_begin → spike_end do
8:             send_gap[i] = sum_sendgap ÷ (spike_end-spike_begin+1)
9:             recv_gap[i] = sum_recvgap ÷ (spike_end-spike_begin+1)
10: function SPIKEREMOVAL
11:     i ← 0, spike_state ← NONE
12:     if recv_gap[0] > recv_gap[1] + SPIKE_DOWN then
13:         spike_begin ← 0
14:         spike_max ← recv_gap[0]
15:         spike_state ← SPIKE_VALID
16:         i ← 1
17:     for i → recv_gap.size()-1 do
18:         switch spike_state do
19:             case NONE
20:                 if recv_gap[i] + SPIKE_UP < recv_gap[i+1] then
21:                     spike_end ← i
22:                     AVERAGEPROBESTREAM(spike_begin,spike_end)
23:                     spike_state ← SPIKE_PENDING
24:                     spike_begin ← i+1
25:                     spike_max ← recv_gap[spike_begin]
26:                 break
27:             case SPIKE_PENDING
28:                 spike_max = max{spike_max, recv_gap[i]}
29:                 if recv_gap[i] + SPIKE_DOWN < spike_max then
30:                     spike_state ← SPIKE_VALID
31:                 else
32:                     break
33:             case SPIKE_VALID
34:                 if recv_gap[i] + SPIKE_UP < recv_gap[i+1] then
35:                     spike_end ← i
36:                     spike_state ← SPIKE_PENDING
37:                     spike_max ← recv_gap[i+1]
38:                 else
39:                     if recv_gap[i]=recv_gap.back() then
40:                         spike_end ← i
41:                     break
42:                 AVERAGEPROBESTREAM(spike_begin, spike_end)
43:                 spike_begin ← i+1
44:     AVERAGEPROBESTREAM(spike_begin, spike_end)
```

Fig. 9: BASS: Pseudo-code

have fairly small gaps ("dips") between them. It is clear that such a bursty buffering event (a "spike" and all the "dips" immediately following it) destroys all patterns in the inter-packet gaps that were present before the event was encountered.

A. Buffering-aware Spike Smoothing (BASS)

The previous section describes existing smoothing techniques that aim to recover patterns in packet delays that last even beyond individual buffering events, by averaging at larger time-scales. Figs 7(a)-(b) illustrate two short probe-streams, however, in which PRC-MT fails to correctly identify whether the probing rate was higher or lower than the available bandwidth. In Fig 7(a), the average of the receive gaps is higher than the send-gap (even though the probing rate is lower



(a) Probing at 6Gbps, BASS (b) Probing at 9Gbps, BASS

Fig. 10: Applying BASS to example probe streams.

than the bottleneck available bandwidth), where in Fig 7(a), it is lower (even though the probing rate is too high). It can be observed that since the spikes and dips are pretty extreme (higher or lower), if the probe-stream does not fall precisely on spike *boundaries*, the receive gaps recovered by these smoothing techniques can be unusually (and inaccurately) low or high.

It follows, naturally, that only buffering events in which all "spikes" and "dips" occur completely within a probe-stream should be considered for averaging out and recovering signatures in the receive gaps. Furthermore, it also suggests that if a smoothing strategy smooths within the boundaries of individual buffering events, rather than smoothing at a fixed timescale that is agnostic of buffering event boundaries, it may better recover the underlying signatures in one-way delays.

Based on the above, we consider the following smoothing strategy to reduce the impact of noise on a probe-stream: (i) explicitly identify the boundaries of each buffering event (starts with a spike and ends with the last dip before the next spike), (ii) eliminate data related to incomplete spikes at either ends of the probe-stream, and (iii) average out the gaps/one-way delays within each buffering event. We refer to this smoothing strategy as *Buffering-aware Spike Smoothing (BASS)*. Fig 9 provides the pseudo-code for BASS.

Fig 10 ilustrates the result when this strategy is applied to the probe streams of Fig 7(a)-(b). After eliminating data from incomplete burst events, the PRC-MT gives correct bandwidth evaluations for both probe streams.

B. Evaluation on Single-rate Probing Framework

We next comprehensively evaluate whether the recovered signature helps improve the performance of the best-performing existing noise-reduction strategy, namely PRC-MT. Fig 11 uses the same probe-streams [7] as Fig 8, and pre-processes each probe-stream using BASS. It then reports the bandwidth decision error for the PRC-MT + BASS combination—we find that when gaps are first smoothened by BASS, PRC-MT performs fairly well, even with probe-stream of just 64 packets. Probes with just 32 packets continue to yield high errors. Note again that each bar in our plots is based

[7]We only provide results here for inter-packet gaps observed at the receiver side. For inter-packet gaps we captured after the bottleneck link, the performance after applying BASS is comparable with using only PRC-MT.

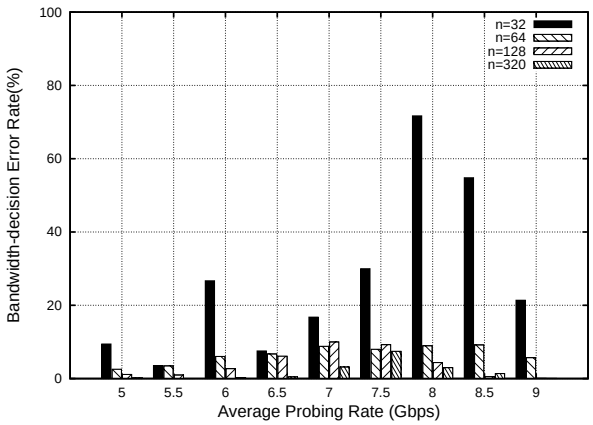


Fig. 11: Bandwidth Estimation Error: PRC-MT + BASS

on results collected from several *thousands* of probe streams, which leads to robust conclusions.

From our analysis in this section, we conclude that short probe-streams can be used effectively at ultra-high speeds when a buffering-event aware smoothing strategy is used to first recover reliable signatures in the packet gaps and delays.

VI. Multi-rate Bandwidth Estimation: Using Buffering-aware Smoothing

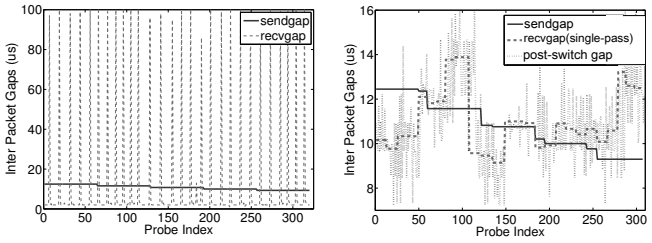
Bandwidth estimation tools differ in the strategy used when probing for multiple candidate rates. The two dominant frameworks are (i) an iterative feedback-based binary-search (adopted by Pathload, PRC-MT, and IMR-Pathload) which probes at a single rate within a probe stream; and (ii) multi-rate non-feedback based probing (adopted by PathChirp and TOPP) which probes for a wide range of candidate rates using back-to-back single-rate probe streams. [26] shows that total probing time can be significantly lowered with the latter strategy, especially on paths with larger round-trip times. It also causes less bandwidth overhead for probing traffic.

The previous section has shown that our buffering-aware smoothing technique significantly reduces probe stream lengths and estimation errors for single-rate probing tools. In this section, we evaluate our smoothing techniques within a multi-rate probing framework.

A. Multi-rate Probing Framework

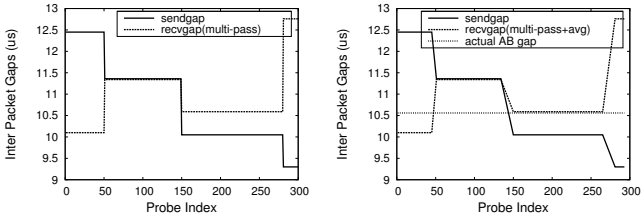
A multi-rate probing framework sends several single-rate probe-streams back-to-back, each at a different probing rate, to create a *multi-rate probe-stream*. Such a framework can be characterized by several parameters: N_r as the number of different probing rates probed by each stream, N_p as the number of packets sent at each probing rate, and $Range$ as the probing range for the whole probe stream.[8] We constrain the the N_r probing rates are uniformly distributed within the $Range$. To estimate bandwidth, the framework finds the highest probing rate at which no self-congestion is experienced—for this, it relies on comparing the receive-gaps to send-gaps and finds the largest probing rate beyond which *all* receive gaps consistently exceed the corresponding sending gaps.

[8]The probing range specifies how much above (or below) the average probing rate is the maximum (or minimum) probing rate adopted within a multi-rate probe-stream—the value of $\frac{Range}{2}$ determines this.



(a) Streams Observed at Receiver (b) Spike Removal

Fig. 12: Applying Spike Removal to Multi-rate Probe Stream



(a) Multi-pass Spike Removal (b) Sliding Window Averaging

Fig. 13: Applying Multi-pass Spike Removal to Multi-rate Probe Stream

In the experiments reported here, we generate multi-rate probe streams by controlling the parameters described above. The control variable for probe stream generation is the average sending rate resulting from the mean of the multiple probing rates in the stream. We control this average rate in fixed increments of 20 Mbps between 1 Gbps and 9 Gbps. The multi-rate streams are generated using the same packet-scheduling Qdisc described earlier. The probe streams share the 10 Gbps link with the highly bursty cross traffic of B-CT. We apply our buffering-aware smoothing techniques discussed in Sec V to each multi-rate probe stream before performing bandwidth estimation as described earlier.

B. Effects of Multi-pass Spike Smoothing

We pick $Range = 30\%$, $N_r = 5$ and $N_p = 64$ to illustrate the effects of *Spike Removal* on typical multi-rate probe streams. Fig 12(a) plots an example multi-rate probe-stream that experiences the typical distortions in receive-gaps due to interrupt coalescence. In Fig 12(b), we illustrate how after applying buffering-aware spike removal, the smoothed receive gaps match the inter-packet gaps experienced by each probe immediately after the bottleneck link. This shows that our smoothing technique effectively mitigates the impact of interrupt coalescence and reveals the time-varying throughput signature of the cross traffic. The smoothed stream, however, still fails to reveal a robust signature of persistent queuing delays because the cross traffic itself is bursty. The remaining (shorter) spikes and dips will lead to considerable estimation

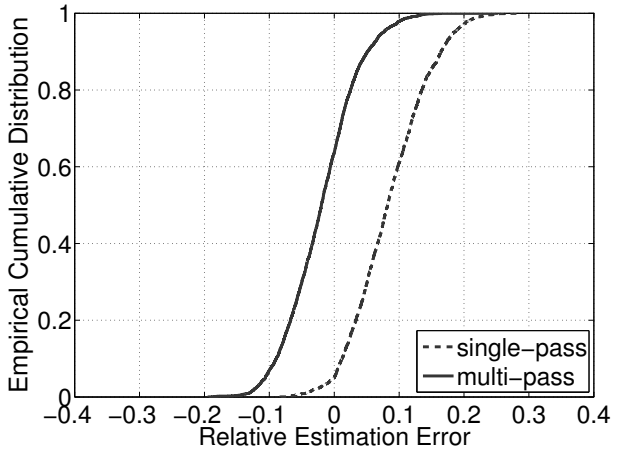


Fig. 14: Multi-pass Spike Removal Reduces Over Estimation

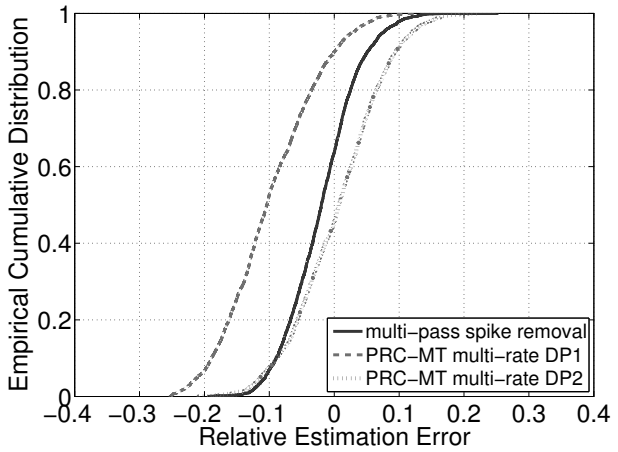
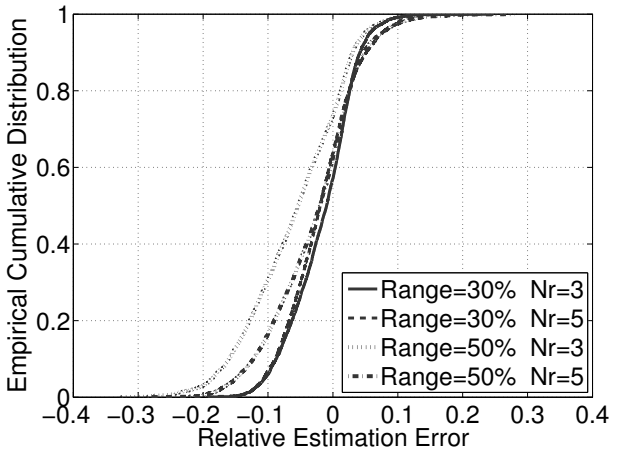


Fig. 15: Multi-pass Spike Removal VS PRC-MT Multi-rate Evaluation

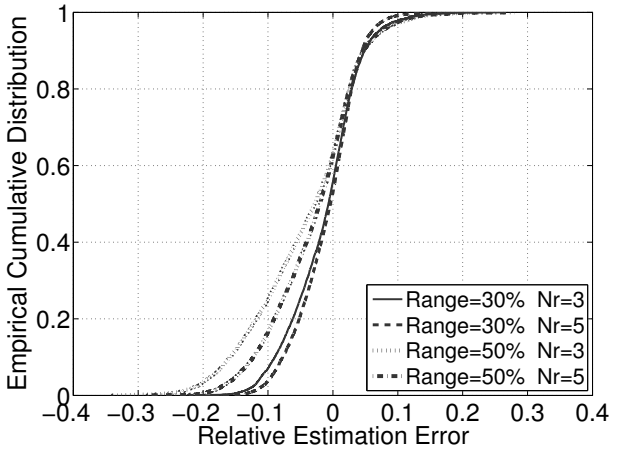
errors. It is natural to ask: will further buffering-aware smoothing of this stream help?

Based on this idea, we develop a *multi-pass spike smoothing* routine which continues to smooth the probe streams until a robust signature of queuing delays is observed. Additionally, we finally apply a sliding window averaging across all smoothed observations with window of size 16. Fig 13 demonstrates the impact of multi-pass spike removal on the same probe stream depicted in Fig 12. The smoothed receive gaps become consistently larger than send gaps after index 145, yielding a clear (and correct) available bandwidth estimate of 6.8 Gbps.

In Fig 14, we plot the distribution of *relative estimation error* observed across all probe streams. If B_g is the ground truth available bandwidth, and B_e is a an estimate of it, then the relative error in bandwidth estimation is given by: $\frac{B_e - B_g}{B_g}$. Fig 14 compares the relative estimation error for single-pass and multi-pass spike smoothing for all probe streams generated in the experiment using the parameters given above. It shows that multi-pass spike removal effectively eliminates much of the severe over-estimation that can result from using only single-pass spike smoothing.



(a) $N_p = 64$



(b) $N_p = 32$

Fig. 16: Multi-pass Spike Removal Estimation Error

C. Sensitivity to Parameters

We performed additional experiments to evaluate the sensitivity of multi-pass spike removal to the parameters of our multi-rate probing framework.

Fig 16(a) plots the estimation error across all probe streams for different probing ranges and number of rates per stream. When *Range=30%*, 95% error is within plus or minus 10% with the median relative error around 0%. Increasing *Range* to 50% allows for a wider range of errors, but 80% of them are still within plus or minus 10%.

To further reduce the probe-traffic overhead, we shrink probe stream length further by decreasing N_p from 64 to 32. From Fig 16(b), we observe that the estimation accuracy is not significantly impacted by probing with fewer packets per rate.

D. Comparison With Multi-rate PRC-MT

In Section V, we showed that, as long as the (single rate) probe stream is no shorter than 64, PRC-MT can be made highly accurate by first smoothing the gaps using BASS. In this section we explore how well PRC-MT + BASS could perform in a multi-rate probing framework.

Within a multi-rate probe stream, those probes sent at the same rate can be treated as a single sub-stream. For each sub-stream, we apply the improved PRC-MT algorithm to produce a binary answer: whether or not the sub-stream rate exceeds avail-bw. Finally the N_r binary answers are evaluated (described below) to pick a probing rate as the bandwidth estimation. We refer to this algorithm as *Multi-rate PRC-MT*.

Notice that cross traffic can be bursty and queues may form and drain completely even within a single sub-stream. If this happens, the estimation decision for *Multi-rate PRC-MT* will be ambiguous. From the probe stream in Fig 12(b), the PRC-MT evaluations for individual rates indicate that the 1st and 3rd rates are below avail-bw, but the 2nd rate exceeds avail-bw. Although the evaluation for each sub-stream is correct, deciding which rate to pick as the better estimation for the whole stream becomes difficult. We investigated two simple decision policies: (DP1) pick the last rate before the first rate above avail-bw (the first rate in this example), or (DP2) pick the last rate below avail-bw (the last rate in the example).

We evaluate these decision methods in experiments with $N_p = 64$, $N_r = 5$, $Range = 30\%$. Fig 15 compares estimation accuracy among DP1 and DP2 policies of *Multi-rate PRC-MT* and *Multi-pass Spike Smoothing*. We observed that the two decision methods produce quite different results for most of the probe streams. DP1 results in consistent under-estimation while DP2 results in consistent over-estimation. Multi-pass spike removal outperforms both PRC-MT multi-rate mechanisms. It shrinks the range of estimation error and is more accurate because of finer granularity in the estimation process.

In this section, we showed how to tailor the buffering-aware smoothing for a multi-rate probing framework. We developed multi-pass spike removal, that automatically smooths the noise caused by interrupt coalescence and bursty cross traffic. We evaluated our de-noising technique with highly bursty cross traffic, showing that a multi-rate probing framework can be successfully scaled to ultra-high speeds.

VII. Application: Ultra-high Speed Congestion Control

Finally, we evaluate the efficacy of our proposed noise-reduction strategy in the context of congestion control for high throughput in long duration flows. A congestion-control protocol aims to probe for the highest sending rate that can be adopted for a sender without causing network congestion. Bandwidth estimation mechanisms have been used in several interesting ways in recent protocols [34], [10]. Of these, TCP Rapid is the one that adopts bandwidth estimation to continuously adapt its sending rate to the available bandwidth along the path to the receiver [10]. It is important to note that past proposals of congestion control protocols that rely on queuing delay estimation have been deprecated because packet delays are often compromised by many sources of noise. In this section, we evaluate whether BASS can help TCP Rapid perform better on real ultra-high speed networks.

f) TCP Rapid: Congestion control in TCP Rapid uses a multi-rate probing framework to continuously perform bandwidth estimation. A TCP Rapid sender transmits all normal outbound data segments organized into logical probe streams of N segments each. The N segments are transmitted in multiple fixed-size sub-streams forming a multi-rate probe stream with exponentially increasing rates in order to probe for several rates within a single probe stream. On receiving ACKs carrying receiver timestamps for each stream of N packets, the sender estimates the available bandwidth for this probe stream. The sender then sets the sending rate (average probing rate) for the next set of data segments transmitted as a probe stream. (Note: our Linux implementation of TCP Rapid extends TCP timestamps to microsecond resolution). The TCP Rapid design aims to ensure that the average load on the path does not exceed its available bandwidth, while also shrinking the congestion control response time.

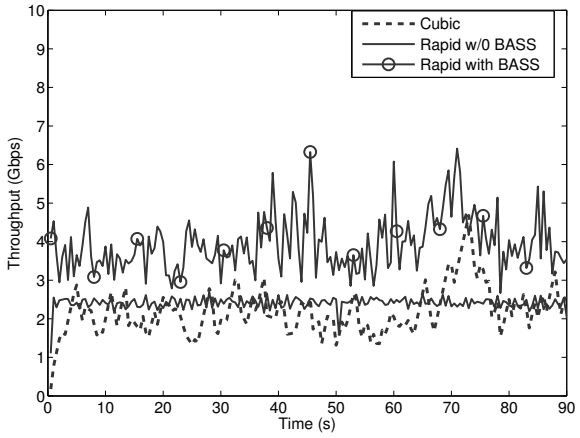
g) TCP Rapid vs. CUBIC: We first compare the throughput performance of TCP Rapid with that of TCP Cubic, which is widely used in Linux distributions. [10] reports the result of *simulation* studies, that Rapid significantly outperforms CUBIC under similar traffic conditions—in this section, we first study if the performance gains also hold in a *real* network environment, in the presence of noise.

These experiments are conducted in the same laboratory testbed described earlier. We use netem (for Cubic) or a locally developed Qdisc based on netem (for Rapid) to emulate RTT by delaying ACKs returned from receiver to the sender. We use a single iperf flow lasting 90 seconds sharing the 10 Gbs link with either the B-CT or S-CT cross traffic and with the RTT for the iperf flow set to 5 milliseconds or 30 milliseconds. For Rapid bandwidth estimation mechanism, we used *Range* = 50%, $N_p = 32$ and N_r=4. Because TCP delayed-ACKs are on by default, only 64 timestamped segments are available to evaluate available bandwidth for each multi-rate probe stream.

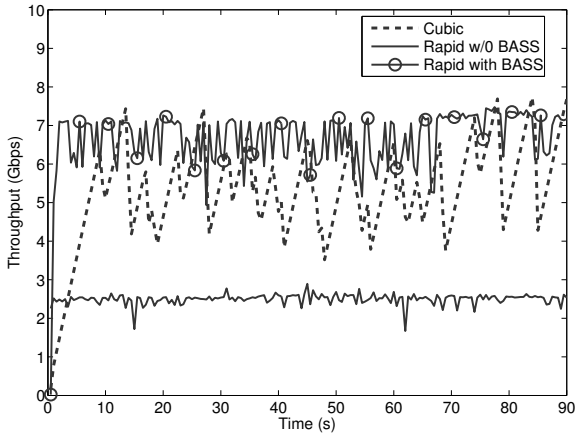
Fig 17 and Fig 18 plot the throughput, observed in 500 millisecond intervals, of the iperf flows using different congestion control mechanisms. The plots seem to suggest that, in general, TCP Rapid achieves higher throughput than CUBIC in the presence of burst cross-traffic B-CT; whereas CUBIC outperforms Rapid in the presence of the smoother S-CT traffic. This is the case for both the small RTT (5 ms) and large RTT (30 ms) scenarios. Upon careful inspection, however, it can be seen that the throughput achieved by Rapid for a given RTT, does not depend at all on the burstiness of the cross-traffic. Furthermore, with both B-CT and S-CT traffic, Rapid throughput is much lower than the network available bandwidth. This suggests that Rapid is unable to estimate well the available bandwidth, and hence, is unable to achieve performance that is even close to that observed in simulation environments [10].

It is also important to note that CUBIC is unable to attain throughput close to the available bandwidth—this is especially true with bursty cross-traffic and on long-RTT paths.

h) BASS + TCP Rapid vs. CUBIC: Next, we use the multi-rate spike smoother, developed in Section VI, as a pre-

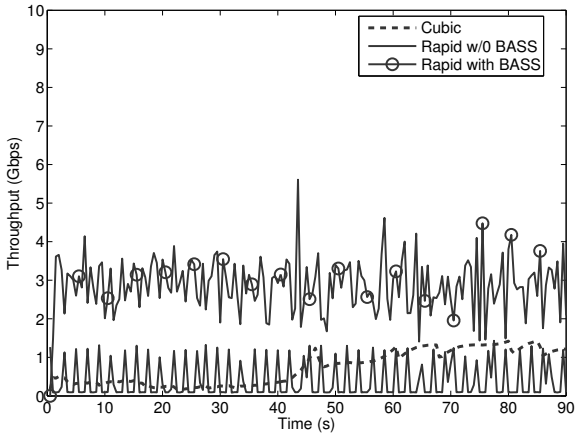


(a) with B-CT

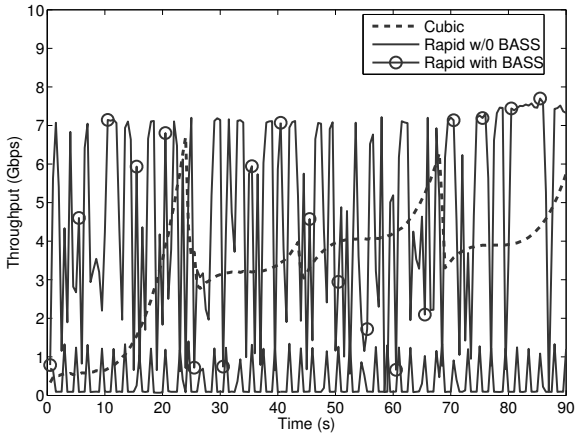


(b) with S-CT

Fig. 17: iperf Throughpput with RTT=5ms



(a) with B-CT



(b) with S-CT

Fig. 18: iperf Throughput with RTT=30ms

cursor to the bandwidth estimator in our Linux implementation of TCP Rapid. The results are included in Fig 17 and Fig 18. We find that in all cases, TCP Rapid yields higher throughput than Cubic. The ratio of throughput improvement (Rapid/Cubic) ranges from 1.35 for the smoothed traffic and 5 millisecond RTT to 3.9 for the bursty traffic and 30 millisecond RTT. While Rapid in some cases has a somewhat higher loss rate than Cubic, it is able to quickly utilize available bandwidth immediately after loss recovery while Cubic requires many RTTs to return to fully utilizing available bandwidth. This is especially noticeable when RTT is 30 milliseconds, and even when the cross traffic is relatively smooth. Most importantly, BASS + TCP Rapid has considerably improved the performance of TCP on 10 Gbps networks.

This is an especially enabling result, since bandwidth-based and delay-based approaches to congestion-control are readily shunned in the research community due to the distrust of packet delays that are corrupted by noise. We believe that our work opens new doors in the area of ultra-high speed bandwidth estimation as well as congestion-control.

VIII. Related Work

The literature is quite rich in the design of bandwidth estimation techniques–we summarize only the most prominent ones here. Probing Rate Model is based on the concept of self-induced congestion as described in Section II. Pathload and Pathchirp are the most well known tools using this model. Pathload [12] sends all packets in a probe stream at the same rate. Each probing rate is evaluated by detecting increasing trends in the one-way delays experienced by probes using two tests (PCT and PDT). It then adopts successive rates in an iterative binary-search manner in order to probe for avail-bw. Another tool, Pathchirp [13], probes avail-bw within a single stream by sending probes at exponentially-increasing rates. Such deliberately designed probing streams help make the tool considerably light-weight [26]. Probing Rate Model is believed to be more robust in the presence of multiple congested links [26], [18].

Although these tools have been shown to perform well in simulation or on low-speed links, their performance in practice on gigabit networks is unpromising [18]. Even on 100Mbps links, noise has been commonly recognized as a

potential issue for bandwidth estimation in practice. [33], [35], [20], [21], [23] all identify timestamping inaccuracy caused by interrupt coalescence as the main challenge to achieving accurate estimation. [35] also points out timer resolution and context-switching overhead as other sources of noise that are limited by system capability. Besides, [36] mentions that ignoring cross traffic burstiness is a major pitfall for existing tools.

Several attempts have been made to deal with noise. In the presence of interrupt coalescence, PathChirp sends a group of packets at each probing rate, and only use the last packet in that group for bandwidth estimation [33]. Pathload adopts a similar strategy as mentioned in Section IV. These strategies are shown to be inaccurate in [20], [21]. IMR-Pathload first applies signal de-noising techniques (window-based smoothing and wavelet transformation) to smooth the noise. Similarly, [22] evaluates the ability of various filters and show that different network scenarios requires different filters.

[23] follows a different track to deal with noise—instead of software-based bandwidth estimation, it seeks hardware assistance. It develops a hardware assistant NIC to precisely control probing gaps and timestamp packet arrivals. The implementation is shown to achieve excellent performance on 10Gbps networks.

In the recent decades, bandwidth estimation mechanisms have been used in several interesting ways in transport protocols. BA-TCP [37] relies on intermediate routers to take measurements of avail-bw and compute the fair share for the satellite TCP connections. Westwood[34] and UDT [11] use avail-bw estimation as guidelines to set congestion window size. Westwood measures avail-bw by tracking ACK receiving rates. UDT sends a probe pair in every N packets and bases its estimation on PGM model. PCT [9] sends small probe streams and estimates avail-bw based on Pathload in slow-start phase and after packet loss in order to better utilize link capacity. TCP Rapid [38] is the only protocol that continuously probes for avail-bw and completely driven by rate-based control.

IX. Concluding Remarks

Bandwidth estimation in high-speed networks has been rejected by many as an infeasible idea due to its sensitivity to noise. In this work, we pursue the ambitious goal of tackling noise even at ultra-high speeds. For this, we use a 10 Gbps testbed to systematically evaluate prior strategies for noise-removal—we showed that some of these work, but only by relying on very long probe streams. We present a new buffering-aware spike removal strategy, which works well even with short probe streams and significantly reduces estimation error in a single-rate probing framework. We then extend this basic mechanism to a multi-pass spike removal strategy and integrate it into a multi-rate probing framework. We show that the combined framework produces accurate estimations with much less network probing overhead. Finally, we illustrate how this enhanced multi-rate probing mechanism is effective in achieving higher throughputs than Cubic when used as the basis for TCP Rapid congestion control, especially on paths with large RTTs.

In future work, we will evaluate our noise-reduction strategy on multi-hop testbeds, and on wide-area high-speed networks. We also plan on evaluating the impact of our strategy in other application domains, in addition to ultra-high speed congestion-control.

References

[1] R.L. Carter and M.E. Crovella. Dynamic server selection using bandwidth probing in wide-area networks. Technical report, Boston, MA, USA, 1996.

[2] R.L. Carter and M.E. Crovella. Server selection using dynamic path characterization in wide-area networks. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1014 –1021 vol.3, April 1997.

[3] S.G. Dykes, K.A. Robbins, and C.L. Jeffery. An empirical evaluation of client-side server selection algorithms. In *Proceedings of IEEE INFOCOM*, volume 3, pages 1361 –1370 vol.3, March 2000.

[4] Y. Zhu, C. Dovrolis, and M. Ammar. Dynamic overlay routing based on available bandwidth estimation: A simulation study. *Computer Networks*, 50:2006, 2006.

[5] X. Zhang, W. Ye, and Y. Jin. Dynamic overlay routing based on active probing measurements: An emulation study. In *Communications and Photonics Conference and Exhibition (ACP)*, pages 1 –2, 2009.

[6] Ren Wang, Giovanni Pau, Kenshin Yamada, M. Y. Sanadidi, and Mario Gerla. Tcp startup performance in large bandwidth delay networks. In *In Proceedings of IEEE INFOCOM*, pages 796–805. Press, 2004.

[7] A. Tripathi and M. Claypool. Improving multimedia streaming with content-aware video scaling. In *Proceedings of IMMCN*, March 2002.

[8] N. Aboobaker, D. Chanady, M. Gerla, and M. Sanadidi. Streaming media congestion control using bandwidth estimation. In *Management of Multimedia on the Internet*, volume 2496 of *Lecture Notes in Computer Science*, pages 89–100. 2002.

[9] 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.

[10] V. Konda and J. Kaur. RAPID: Shrinking the Congestion-control Timescale. In *Proceedings of the IEEE INFOCOM*, April 2009.

[11] Yunhong Gu. *UDT: A High Performance Data Transport Protocol*. PhD thesis, Chicago, IL, USA, 2005. Chairperson-Robert L. Grossman.

[12] Manish Jain and Constantinos Dovrolis. Pathload: A measurement tool for end-to-end available bandwidth. In *In Proceedings of Passive and Active Measurements (PAM) Workshop*. Citeseer, 2002.

[13] Vinay Ribeiro, Rudolf Riedi, Richard Baraniuk, Jiri Navratil, and Les Cottrell. pathchirp: Efficient available bandwidth estimation for network paths. In *Passive and active measurement workshop*, volume 4, 2003.

[14] J. Navratil. ABwE: A Practical Approach to Available Bandwidth. In *Proceedings of PAM*, 2003.

[15] C. Dovrolis and M. Jain. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. *IEEE/ACM Transactions in Networking*, August 2003.

[16] Albert Cabellos-Aparicio, Francisco J Garcia, and Jordi Domingo-Pascual. A novel available bandwidth estimation and tracking algorithm. In *Network Operations and Management Symposium Workshops, 2008. NOMS Workshops 2008. IEEE*, pages 87–94. IEEE, 2008.

[17] Google fiber project. https://fiber.google.com/.

[18] A. Shriram, M. Murray, Y. Hyun, N. Brownlee, A. Broido, M. Fomenkov, and K.C. Claffy. Comparison of public end-to-end bandwidth estimation tools on high-speed links. In *PAM*, pages 306–320, 2005.

[19] Qianwen Yin, Jasleen Kaur, and F Donelson Smith. Scaling bandwidth estimation to high speed networks. In *Passive and Active Measurement*, pages 258–261. Springer, 2014.

[20] Seong-Ryong Kang and Dmitri Loguinov. Imr-pathload: Robust available bandwidth estimation under end-host interrupt delay. In *Passive and Active Network Measurement*, pages 172–181. Springer, 2008.

[21] Seong-Ryong Kang and Dmitri Loguinov. Characterizing tight-link bandwidth of multi-hop paths using probing response curves. In *Quality of Service (IWQoS), 2010 18th International Workshop on*, pages 1–9. IEEE, 2010.

[22] Uyen C Nguyen, Dung T Tran, and Giang V Nguyen. A taxonomy of applying filter techniques to improve the available bandwidth estimations. In *Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication*, page 18. ACM, 2014.
[23] Akeo Masuda, Akinori Isogai, Kohei Shiomoto, Yoshihiro Nakajima, Tetsuo Kawano, and Mitsuru Maruyama. Application-network collaborative bandwidth on-demand for uncompressed hdtv transmission in ip-optical networks. In *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pages 177–180. IEEE, 2010.
[24] N. Hu and P. Steenkiste. Evaluation and Characterization of Available Bandwidth Probing Techniques. *IEEE JSAC Internet and WWW Measurement, Mapping, and Modeling*, 2003.
[25] Jacob Strauss, Dina Katabi, and Frans Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proceedings of the ACM SIGCOMM Internet Measurement Conference '03*, Miami, Florida, October 2003.
[26] A. Shriram and J. Kaur. Empirical evaluations of techniques for measuring available bandwidth. In *Proceedings of IEEE INFOCOM 2007*, May 2007.
[27] M. Jain and C. Dovrolis. Pathload: an available bandwidth estimation tool. In *PAM*, 2002.
[28] 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.
[29] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Global Internet Symposium*, November 2000.
[30] Paul Barford and Mark Crovella. Generating representative web workloads for network and server performance evaluation. *ACM SIGMETRICS Performance Evaluation Review*, 26(1):151–160, 1998.
[31] Aaron Turner and Matt Bing. Tcpreplay, 2005.
[32] Jeffrey S Chase, Andrew J Gallatin, and Kenneth G Yocum. End system optimizations for high-speed tcp. *Communications Magazine, IEEE*, 39(4):68–74, 2001.
[33] Ravi Prasad, Manish Jain, and Constantinos Dovrolis. Effects of interrupt coalescence on network measurements. *Passive and active network measurement*, pages 247–256, 2004.
[34] Saverio Mascolo, Claudio Casetti, Mario Gerla, Medy Y Sanadidi, and Ren Wang. Tcp westwood: Bandwidth estimation for enhanced transport over wireless links. In *Proceedings of the 7th annual international conference on Mobile computing and networking*, pages 287–297. ACM, 2001.
[35] Guojun Jin and Brian L Tierney. System capability effects on algorithms for network bandwidth measurement. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 27–38. ACM, 2003.
[36] Manish Jain and Constantinos Dovrolis. Ten fallacies and pitfalls on end-to-end available bandwidth estimation. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 272–277. ACM, 2004.
[37] Mario Gerla, Wenjie Weng, and R Lo Cigno. Ba-tcp: A bandwidth aware tcp for satellite networks. In *Computer Communications and Networks, 1999. Proceedings. Eight International Conference on*, pages 204–207. IEEE, 1999.
[38] Vishnu Konda and Jasleen Kaur. Rapid: Shrinking the congestion-control timescale. In *INFOCOM 2009, IEEE*, pages 1–9. IEEE, 2009.

APPENDIX A
EXPERIMENTAL TESTBED

a) Testbed Hosts: The core of the network illustrated in Fig 1 consists of two HP 2900 switches with multiple 1 Gbps and 10 Gbps ports, both copper and fiber. The switch-to-switch path is a 10 Gbps fiber path. The host that is used to generate probe streams is a Dell PowerEdge R720 with four cores (hyperthreads for 8 logical processors) running at 3.3 GHz. The 10 Gbps Ethernet adapter is a PCI Express x8 Myricom 10 Gbps copper NIC with the myri10ge driver. The Endace DAG monitor NICs are hosted on high-end Dell servers with very large memory buffers. It provide line-rate capture of all frames traversing the switch-to-switch path along with timestamps of nanosecond precision and 10 nanosecond accuracy.

The host used to receive the probe-streams is also a Dell PowerEdge R720 running at 3.3 GHz. The 10 Gbps Ethernet adapter is a PCI Express x8 Intel X520 fiber NIC with the ixgbe driver. The ethtool program was used to turn off segment and receive offload functions on all the NICs used to send and receive probe streams. Experiments were run with the default receive coalesce interval any adaptive capability.

The network includes an additional 10 pairs of hosts (sender and receiver) that are used to generate cross traffic sharing the switch-to-switch link. The CPU speeds of these hosts range from 1.8 GHz to 3.3 GHz, all have 1 Gbps copper NICs. All hosts in the network run recent release of RedHat Enterprise Linux 6 with the Linux kernel 2.6.32.

b) Send-gap Creation: Within each probe stream, the Qdisc computes the intended departure time (in nanoseconds) for each frame in a probe stream and runs a scheduling algorithm that ensures that the frame is dequeued to the device driver no earlier than the intended departure time. A log was generated by the kernel module (with printk) with sufficient information to determine the sequence number and intended send gap for all frames in a probe stream.

In order to eliminate most of the uncontrolled noise in the send gaps, the frame-to-frame gaps are enforced in the sending NIC queues by inserting one or more Ethernet PAUSE frames with appropriate sizes between frames in a probe stream. These control frames are specified as part of the IEEE 802.3x for flow control between two ends of a link. PAUSE frames are discarded by a receiving switch and thus consume bandwidth only on the NIC to switch link. As a result, the intended inter-packet gaps are preserved between successive frames arriving at the first outbound queue. We evaluate the inter-packet gaps generated this way by comparing against those recorded at the DAG monitor when the path is free of cross traffic, showing that 90% of them are within 1μs difference with the intended gaps.

c) Receiver-side Timestamping: Ideally, gaps between frames arriving at the receiver would be computed from timestamps taken in hardware by the NIC at the arrival of each frame using a high-resolution clock local to the adapter. Unfortunately the vast majority of 10 Gbps adapters currently available only do hardware timestamping for PTP frames (if they do timestamping at all). This means that timestamps used for measuring receiver gaps must be done in software.

Existing bandwidth estimation tools timestamp packets for measuring receiver gaps in software at the application layer. In order to record software timestamps with the best-possible accuracy, we implement a kernel loadable module attached as an ingress Qdisc to the adapter that is receiving frames from the switch. The position of this ingress Qdisc in the Linux networking stack is logically between the device driver and the bottom of IP. The function of this module is basically to record and log a kernel-generated (ktime_get()) timestamp with microsecond precision.