Making a Delay-based Protocol Adaptive to Heterogeneous Environments

Kiran Kotla and A. L. Narasimha Reddy
Texas A&M University
{kiran@cs, reddy@ece}.tamu.edu

Abstract— **This paper investigates the issues in making a delay-based protocol adaptive to heterogeneous environments. We address how a delay-based protocol can compete with a loss-based protocol such as TCP. We investigate if potential noise and variability in delay measurements in environments such as cable and ADSL access networks impact the protocol behavior significantly. We investigate these issues in the context of incremental deployment of a new delay-based protocol, PERT. We propose design modifications to PERT to compete with SACK. We show that PERT experiences lower drop rates than SACK and leads to lower overall drop rates with different mixes of PERT and SACK protocols. Second, we show that a single PERT flow can fully utilize a high-speed, high-delay link. The results from ns-2 simulations indicate that PERT can adapt to heterogeneous networks and can operate well in an environment of heterogeneous protocols. We also show that proposed changes retain the desirable properties of PERT such as low loss rates and fairness, when operating alone. The protocol has also been implemented in the Linux kernel and tested through experiments on live networks, by measuring the throughput and losses between nodes in our lab at TAMU and different machines on the planet-lab .**

*Keywords-**Congestion protocol, delay-based, ns-2 simulations, emulations, incremental deployment , high-speed links***

I. INTRODUCTION

Multimedia applications have low-delay and low-loss requirements. Currently, applications either use UDP or TCP as transport protocols. UDP does not provide congestion control and applications employing UDP have to incorporate congestion control. TCP's in-order reliable delivery may not be necessary for multimedia applications. A number of congestion control approaches have been proposed to replace TCP to move networks towards higher multimedia content and higher link bandwidths in the future.

Traditionally, congestion protocols have taken two different approaches of inferring congestion from network feedback – (a) through packet losses or markings at the router, or (b) from measured characteristics such as delays at end hosts. Challenges in estimating delays accurately and other issues have resulted in skepticism of viability of delay-based schemes [1], [2]. Recently, we addressed some of these issues and proposed a delay-based protocol, PERT (Probabilistic Early Response TCP) [3]. PERT improves the delay estimation process and deals with remaining uncertainties through a probabilistic response to measured delays. PERT emulates AQM's behavior at end hosts, responding at a higher rate at higher delays. While PERT has been shown to be effective in reaching its goals, a number of technical challenges remain in its practical deployment.

First and foremost is the issue of how delay-based protocols can compete with various versions of TCP. Delay-based protocols, by responding to congestion early, cede ground to loss-based protocols that keep increasing their rate until a packet is dropped (as most versions of TCP do). While most delay-based protocols exhibit good properties in a homogenous deployment, for the delay-based protocols to be practical, they need to be able to operate in an environment of mixed protocol deployment. This would be necessary for incremental deployment.

We address this issue (and others) in this paper. We propose and evaluate design enhancements to enable the delay-based protocol, PERT to compete with TCP-SACK. We also study what advantages and benefits may be realized as a mix of PERT and SACK flows evolves from 100% SACK to 100% PERT. We show through these experiments that incremental deployment of PERT provides lower overall drop rates along with lower drop rates for the PERT flows. While some recent protocols [5, 18, 21] have strived for coexistence with TCP, we are not aware of any work that simultaneously deals with incremental deployability of a new protocol and fair bandwidth sharing with TCP in mixed protocol deployment workloads.

A second issue that has been raised in the past regarding delay-based protocols is their robustness to noise in delay measurements. This has been the primary motivation for employing the probabilistic response in PERT [3]. Recent work on cable and ADSL access networks has highlighted the RTT variance of these networks even in the absence of congestion [17], due to their network access granting and scheduling mechanisms. This raises the question whether delay-based congestion protocols can function effectively when deployed in such access networks.

We study this issue through practical deployment of a delay-based protocol in cable and ADSL networks. We report on PERT's ability to correctly gauge congestion in networks with widely varying access delays. We also evaluate PERT's robustness to measurement noise by deliberately adding noise to measured delays in simulations. These evaluations show that PERT can be more robust to noise in delay compared to FAST and Vegas.

A third issue we address is whether a delay-based protocol can be scaled to provide high utilizations for single flows in high-speed, high-delay links. This has been a topic of considerable interest lately and many new protocols have been proposed [4, 5, 6, 7, 8]. We show that PERT can fully utilize high-speed, high-delay network links and provide much lower drop rates than loss-based schemes such as [4, 6, 7].

While FAST [5], a delay-based protocol, has been designed to compete with TCP and operate in high-speed networks, not much work has been reported on its performance in environments of mixed protocol deployment. Our work here emphasizes this aspect. We compare PERT's performance with that of FAST in incremental deployment scenarios.

We propose, analyze and evaluate design modifications for PERT to deal with these important issues. The suggested modifications are simple to implement and are shown to be effective through analysis, ns-2 based simulations and testing our Linux based kernel implementation over the Internet.

The paper makes the following significant contributions: (a) Adapted a delay based protocol to be competitive with the existing loss based congestion protocol TCP, (b) conducted extensive evaluations of mixed deployment scenarios of PERT and TCP to show that PERT offers benefits to individual adopters while providing incremental benefits to network characteristics and hence providing a path to incremental deployment. (c) Adapted PERT to high speed networks to enable a single flow to utilize a link fully with zero losses and (d) retained the properties of the original PERT when operating alone in homogenous environments (and providing nearly zero packet losses and low queue lengths).

The rest of the paper is organized as follows. In Section 2 we give a brief outline of PERT to keep the paper self-contained. In Section 3 we propose two modifications to PERT. In Section 4, we present results from extensive ns-2 simulations and in section 5 we conclude the paper with a discussion of remaining open issues in adopting PERT.

II. BRIEF BACKGROUND ON PERT

PERT uses smoothed RTT measurements to decipher the state of congestion in the flow's path. Like other delay-based protocols, PERT presumes that the path is congested if the observed delay is higher. However, PERT recognizes the uncertainty in congestion prediction due to noise in measurements and burstiness of traffic. In order to mitigate these effects, PERT employs a probabilistic response to measured delays. PERT reduces the impact of false positives (in congestion prediction), by using a smaller probability of response when the perceived queue length is small, and a larger probability of response when the perceived queue length increases.

While PERT can be designed to emulate any AQM mechanism, we will focus our attention here on a version of PERT that emulates RED. The probabilistic response of PERT is designed to be similar to that of RED. Fig. 1 shows the probability of response against the congestion detection signal, smoothed RTT. Similar to RED, we define two thresholds *minthresh_* and *maxthresh_* and the maximum probability of response *maxP_*. When the value of $srtt_{0.99}$ is below the *minthresh_* the probability of response is 0. As the value of $srtt_{0.99}$ increases beyond *minthresh_*, the probability of reducing a window in response to each ack linearly increases until it reaches the value *maxP_* at *maxthresh_*. Similar to the "gentle" variant of RED, between *maxthresh_* and (2 * *maxthresh_*), the probability increases between *maxP_* and 1. Beyond (2 * *maxthresh_*), the probability remains constant at 1[1]. For the parameters *minthresh_*, *maxthresh_* and *maxP_* we use fixed values of (5ms, 10ms and 0.05) respectively. It is possible to choose these values adaptively based on network conditions similar to the mechanisms suggested in [9].[1]

When queue lengths are observed to be above thresholds, every ack arrival indicates congestion until the queue lengths

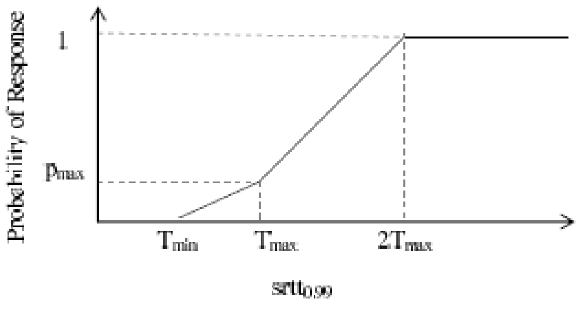


Figure 1: Probabilistic response curve used by PERT.

fall below the thresholds. It is not necessary for the flow to respond to each of these indications. The impact of response may not be seen until after an RTT. Hence, the early response to congestion is limited to once per RTT (even when random probability may pick multiple packets in one RTT)

III. DESIGN ENHANCEMENTS TO PERT

First, in order to address the issue of PERT's ability to compete with TCP, we carry out a steady state throughput analysis of PERT. For a flow with window adjustment of $W = W + \alpha / W$ on an ack and $W = (1-\beta) * W$ on a congestion response, the steady state throughput of the flow is given by $\frac{1}{RTT} * \sqrt{(\frac{\alpha}{\beta * p})}$ [10]. PERT's response to congestion can be broken into two probabilities p' and p, where p' corresponds to the early response probability and p corresponds to the observed congestion loss probability. When PERT competes with SACK, SACK only observes the congestion loss probability. It has been observed that the congestion loss probabilities of different protocols may be different when they compete with each other [11], but for simplicity of analysis, we assume that SACK and PERT observe similar congestion loss probability p (which is validated through simulations later in the paper).

PERT's throughput is controlled by the combined early and congestion response probabilities and is given by $1-(1-p)*(1-p') = p + p' - p * p'$. If PERT has to roughly get an equal share when competing with TCP, comparing the steady

[1] Different response functions can be chosen. "Gentle" RED is used here as a representative function

state throughput equations of the two protocols [11], we will need:
$\beta_{PERT} * (p + p' - p * p') / \alpha_{PERT} = \beta_{TCP} * p / \alpha_{TCP}$

Since $\alpha_{TCP} = 1$, we get:
$\alpha_{PERT} = \beta_{PERT} * (p + p' - p * p') / (\beta_{TCP} * p)$.

Conservatively, we set $\beta_{PERT} = \beta_{TCP}$ to get:
$\alpha_{PERT} = p + p' - p * p' / p \approx 1 + p' / p$.

This steady-state throughput analysis gives us an idea of how aggressive PERT needs to be in increasing the window in order to counter its early response behavior for it to get an equal share of link throughput when competing with TCP. The first design modification we make is based on this analysis. We make PERT's window $W = W + \alpha_{PERT}$, where $\alpha_{PERT} = 1 + p'/p$.

When the proactive congestion response is successful, the queue lengths are expected to be maintained low. As a result, it is not necessary to respond with a 50% window reduction in case of early response. In [12] the authors show that the router buffers are set to the delay-bandwidth product of the link since the TCP flow reduces its window by 50%. If the TCP flow were to use a factor β instead for window reduction, then the relationship between the buffers and the window reduction factor can be re-written as $B > \frac{\beta}{1-\beta} * BDP$.

When a PERT flow responds to congestion early, it takes the link a smaller amount of time to flush the packets in the buffer than when the packet is dropped when the buffer is full. As a result, PERT should be less aggressive in reducing its rate if we want to keep the link utilization high with a single flow. We follow the analysis of [12] and modify PERT's early response to reduce the window by *cur_qdelay/(cur_qdelay + max_qdelay)*, where *cur_qdelay* is the estimated queuing delay at the time of early response and *max_qdelay* is the maximum observed queueing delay. It is observed that when *cur_qdelay* = *max_qdelay*, the window is reduced by a factor of 0.5, as is the case when a packet is dropped. The window reduction factor now varies from 0 to 0.5 depending on the observed queuing delay. As a result, the actual value of β_{PERT} would depend on the relative ratio of early response rate p' and the packet loss rate p. It is noted that we assumed that β_{PERT} conservatively to be the same as that of β_{TCP} earlier. We will evaluate PERT through simulations and live experiments to observe that this assumption does not significantly affect its fairness.

The local stability of PERT under these changes has been analyzed [13]. The analysis shows that these changes actually improve the local stability region of PERT [13]. Due to space constraints, we omit the details of analysis here and refer the reader to [13].

An important consideration is to decide when PERT is operating in a homogenous environment (with all the flows being PERT) or a heterogeneous environment with competing TCP flows. While the aggressive window increase will make PERT competitive against TCP in a mixed environment, it may increase the packet losses and queue lengths in a homogenous environment. We use the observed queuing delays to guide the window increase function.

When observed queue delays are high, PERT assumes that it is competing with a loss-based protocol such as TCP that tends to push queue lengths high. Hence, it needs to be aggressive in increasing its window to compensate for its early response. We use a threshold of 0.5*max_observed queuing delay to conclude that PERT is competing against TCP. Similarly, when queuing delay persistently stays below the min_thresh, PERT concludes that it is operating in an environment where link bandwidth is plenty. In these two modes, PERT increases its window increase factor α from a default of 1 to a higher value. When queuing delay is higher, its increase is guided by a desire to compete with loss-based protocols (hence up to 1+p'/p) and when queuing delay is low, its increase is guided by a need to fill the link bandwidth. We can see from Fig. 1 that the third threshold was set to 2**maxthresh_* in the original PERT. We now adjust it dynamically with the maximum observed queue length as 0.65*max_observed queuing delay.

Essentially, now PERT operates in three different phases. When the observed queuing delay is less than min_thresh (5ms, in this paper), it deciphers that the bandwidth is being underutilized and increments α linearly till it reaches a threshold of 32, to increase its window during the congestion avoidance phase. When the queuing delay is greater than min_thresh, but less than half the maximum observed queue length, it deciphers that it is utilizing the available bandwidth and since the observed queuing delay is relatively small, it assumes that all the competing flows are of the PERT flavor and decrements α till it reaches 1. However, if the observed queuing delay is larger than half the queue length, it deciphers that there are some flows which use a loss based congestion response function and increments α till it reaches α_{PERT}. This adaptation of α takes place at slightly longer time scales than an RTT (we used 5*RTT in our implementation). The target alpha is smoothly varied as the buffer transitions from one region to another region.

3.1 Implementation Issues

We implemented the above modifications to PERT in ns-2. We implemented the TFRC algorithm for estimating the packet drop rate p [14]. We employed a similar method for estimating the early response rate of PERT p'. When PERT is successful in curtailing packet drops to very small number or to zero, the alpha parameters computed by $1 + p'/p$ either becomes too large or infinity. Too large an alpha value is not practical and can result in excessive burstiness and consequent problems of packet drops and higher queue lengths. Hence, we set $\alpha_{PERT} = \min(\alpha_{\max}, 1 + p'/p)$ where $\alpha_{\max}$ is a parameter chosen to control this burstiness. In our simulations and emulations below, we chose $\alpha_{\max}$ to equal 32. It seems feasible to make this parameter dependent on both the observed drop rate and observed queuing delays, which we will explore in the future.

When a single PERT flow starts on a link, it has no idea of the maximum queuing delay (or the size of the buffer). This is necessary to implement the second suggested modification above. We initialize maximum queuing delay to maximum queuing delay threshold utilized in PERT. If maximum queuing delay is initialized to zero, since PERT's early response keeps the queuing delays low, PERT's window reduction stays close to 0.5 and could lead to under utilization of the link in single flow situations.

We evaluate PERT in the following section in different traffic scenarios.

IV. EXPERIMENTAL EVALUATION

We have conducted extensive ns-2 based simulations and Linux kernel based experiments to evaluate PERT. We attempt to make our evaluation realistic by simulating a wide range of network parameters, not all the results are presented here due to space constraints. We present the results for a single bottleneck topology. For all the experiments, the bottleneck buffer size is set to the bandwidth-delay product, unless otherwise mentioned, with the minimum number of packets being equal to at least twice the number of flows. All simulations are run for 400 seconds and reported results are measured during the stable period between 100 and 300 seconds. When multiple flows share a link, their start times are chosen randomly in the range (0, 10) seconds to avoid synchronization.

4.1 Varying Mix

In this experiment, the bottleneck link bandwidth is kept constant at 150Mbps. The end-to-end RTT of the flows is set to 60ms and the total number of flows is set to 100. For the first two experiments, the percentage of PERT flows is varied from 0 to 100. Fig.2 summarizes the results. We see that the normalized queue length does not vary significantly when there is a mix of PERT and SACK flows. However, when all the flows are of PERT, a significant drop in the queue length is observed. The drop rate graph shows that the drop rate of the mix reduces, though not significantly as the share of PERT flows increases. The drop rate goes to zero when the flows are 100% PERT.

For the following experiments, the percentage of PERT flows is varied from 5 to 95, because drop ratio is defined only for the cases where we have both PERT and SACK flows. Fig.3 shows the PERT to SACK drop ratio in the mixed environment. We see that this ratio is always less than ‘1’, meaning PERT always has a lower drop rate in the mix. The second graph in Fig.3 shows the percentage of PERT’s bandwidth share as the share of PERT flows increases. This shows that the observed share is almost similar to the expected (fair) bandwidth share.

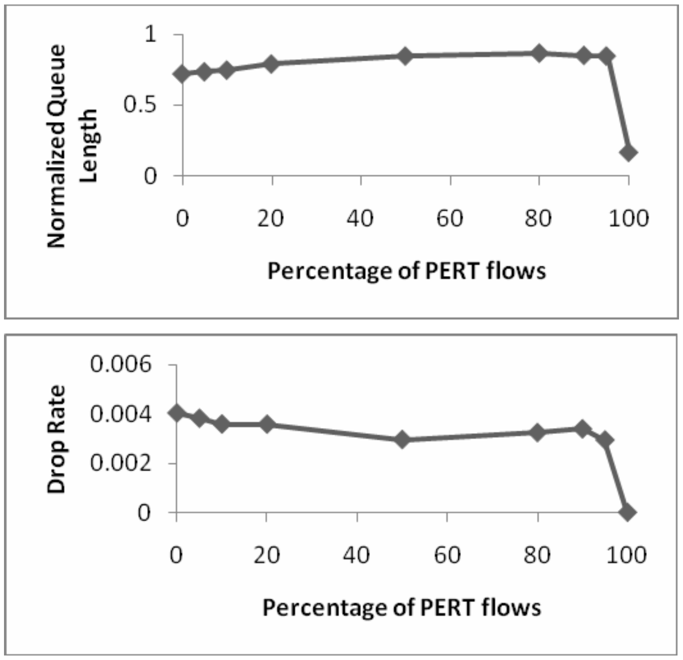


Figure 2: Variation of drop rate and normalized queue length with percentage of PERT flows

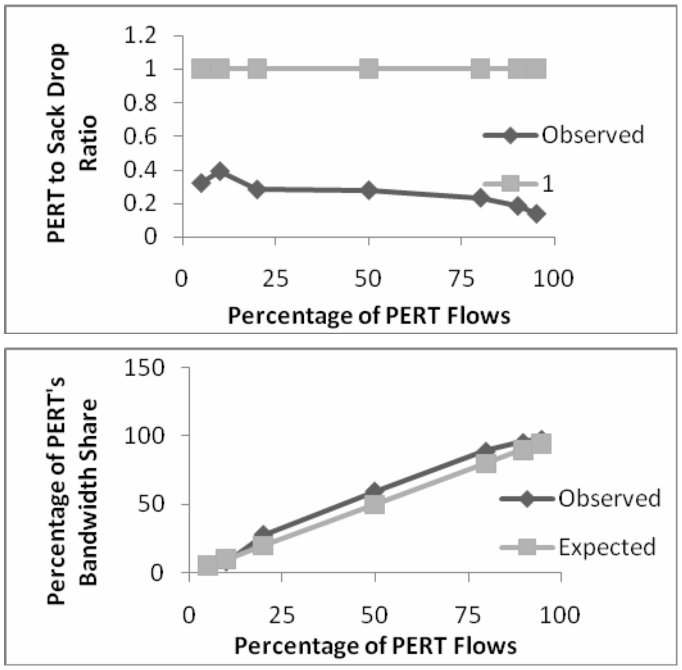


Figure 3: Variation of PERT’S bandwidth share and FAST to SACK drop ratio with percentage of PERT flows.

In the same experiment, we calculated the mean and variance of the bandwidth (measured once in a second) for PERT and SACK flows. Fig.4 shows the ratio of variance and mean of the bandwidth for PERT and SACK. We can see that, PERT maintains a lower variance per mean even in the mix scenarios and the variance keeps reducing as the percentage of the PERT flows increases in the mix of the flows. This result also points to the potential benefits of incremental deployment of PERT.

Results in Fig.2 and Fig.3 show the feasibility of the incremental deployment of PERT. As the mix of protocols goes from 100% SACK to 100% PERT, PERT can coexist with TCP, sharing bandwidth nearly fairly. PERT flows can benefit from lower drop ratios in the mixed environment, giving an incentive for the adoption of PERT over TCP (we point out that bandwidth incentives are easy to provide compared to the gains observed here in packet loss rates). These results show that PERT's deployment benefits the individuals deploying PERT in an environment of mixed protocols while enjoying the benefits of nearly zero packet losses and low queue lengths in

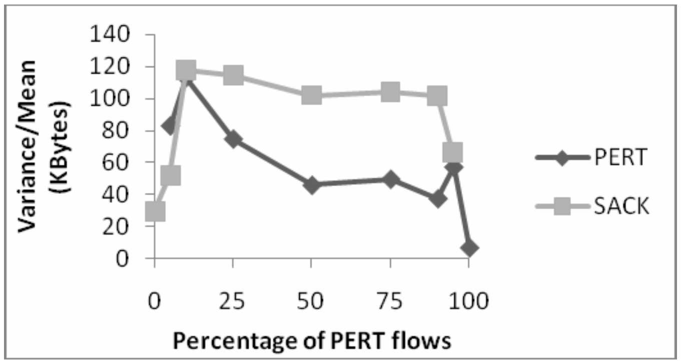


Figure 4: Variation of the Ratio Variance/Mean of PERT'S bandwidth with percentage of PERT flows.

homogenous environments. We also observe that the drop rate is lower in a mixed deployment environment than with 100% SACK flows. We plan to explore more techniques for improving network characteristics globally, rather than just for PERT flows, in the future.

An experiment with a similar setup is repeated for a mix of FAST [5] and SACK. Fig 5 summarizes the results. From Fig. 5, we see that though the queue lengths decrease with the increase in the percentage of FAST flows, and FAST maintains lower drop rate compared to that of SACK in a mixed scenario, the overall drop rate increases with the increase in the percentage of FAST flows. Further, in the mixed environment, FAST gets uneven (and much larger) share of bandwidth compared to SACK. These results indicate that FAST's modifications for competing against TCP may not beneficial to both TCP and itself (results in higher drop rates) even in a 100% FAST environment.

Due to lack of space, we don't explore FAST's performance much further in this paper.

4.2 Variation of number of flows in a 50-50 Mix of PERT and SACK

In this experiment, the bottleneck link bandwidth is set to 150Mbps and the total number of long-term flows is varied from 20 to 1000, with 50% of PERT and 50% of SACK flows in each case. The end-to-end RTT is 60ms. Fig. 6 shows the results. As expected, we see that the queue length and drop rate increase with the number of flows. We also see from Fig. 6 that the bandwidth share of PERT is low when there is less number of flows sharing the bandwidth and that it raises and stabilizes at 50% as the number of flows increases. This is because PERT

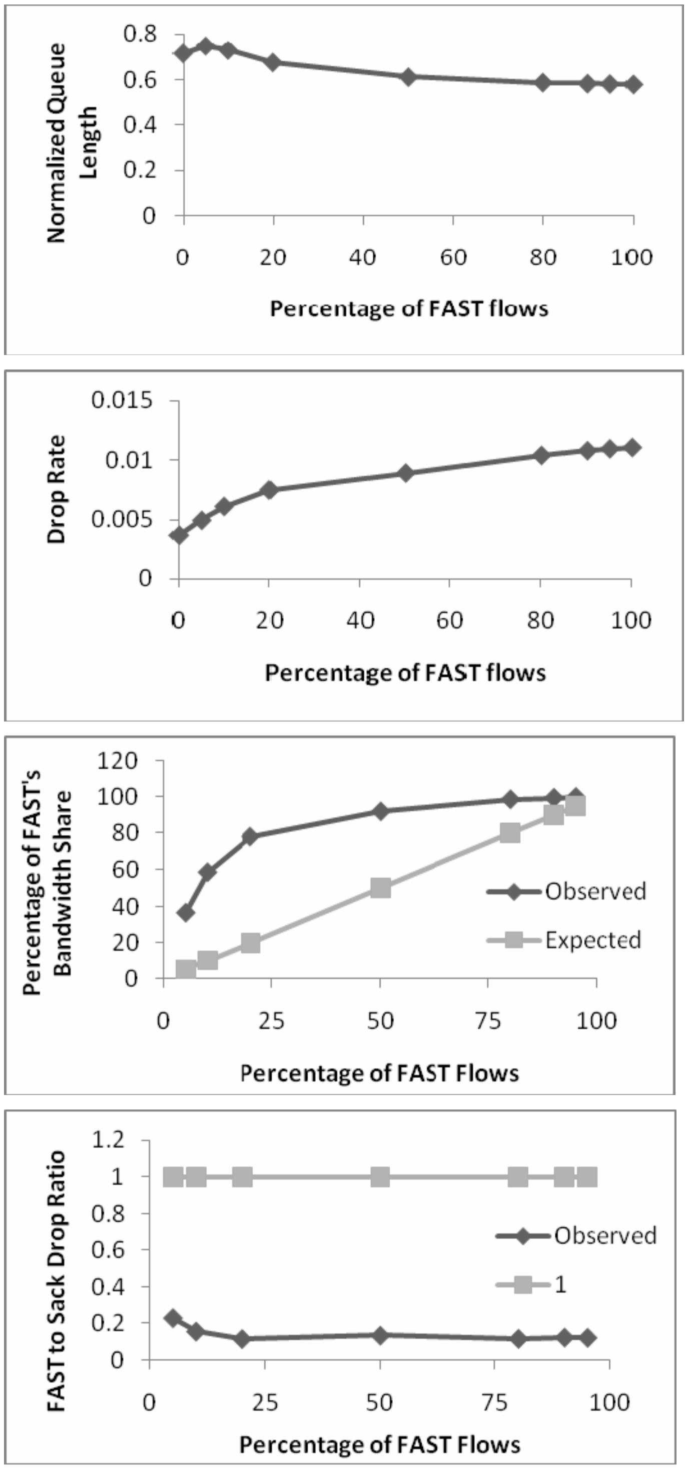


Figure 5: Variation of Normalized Queue Length, Overall drop rate, FAST'S bandwidth share and FAST to SACK drop ratio with percentage of PERT flows.

does not operate in the aggressive (Compete with TCP) mode all the time, when the available bandwidth is high and the queue length stays below half the maximum queuing delay. We also see that the Jain's Fairness index varies accordingly as PERT's bandwidth share.

4.3 Impact of Bottleneck Link Buffer Size in a 50-50 Mix of PERT and SACK

In this experiment, the bottleneck link bandwidth is kept constant at 150Mbps. The end-to-end RTT of the flows is set to 60ms and the number of flows is set to 80, with 40 flows of PERT and 40 flows of SACK. The buffer size at the bottleneck is varied as a factor of Bandwidth-delay product from 0.25 to 4. With this setup PERT and SACK are compared. Fig. 7 shows the PERT to SACK Drop Ratio and the observed PERT's Bandwidth share. We see that the PERT's Bandwidth share is really high when the bottleneck buffer length is small and it loses to SACK only when there is a very large buffer. This is because, at lower bottleneck buffer sizes, the queue length gets larger than half the maximum length at an earlier stage and PERT operates in aggressive (Compete) mode. We also see that PERT to SACK drop ratio stays below '1' in all cases. This shows that PERT performs very well with small buffers. PERT loses to SACK in the presence of larger buffers (3.5x BDP and above) because PERT operates in the safe region and tries to reduce the queue length, while SACK does the opposite. Moreover, it is observed that PERT stays fair to TCP over a large range of buffer sizes, ranging from 0.5xBDP to 3xBDP.

These results provide a further incentive for the deployment of PERT as the need for using small buffers has been highlighted by several studies, both from the vantage of building faster routers and from multimedia applications demand for small delays.

4.4 Performance in High-speed networks

Now that PERT performs well in a mixed flow scenario, we have performed tests to see how PERT performs in high-speed networks with large available bandwidths. In this experiment, the bottleneck link bandwidth is kept constant at 2.4Gbps. The end-host link bandwidth is varied from 10Mbps to 2.4 Gbps. The end-to-end RTT of the flows is set to 70ms and the experiment is run with a single flow. Fig 8 summarizes the results. It is observed that a single PERT flow can nearly fully utilize a high speed link with zero packet losses. This may be compared to the recent proposals for high-speed protocols [4, 6, 7] which show significantly higher packet losses (several orders of magnitude difference) as shown in [4].

4.5 Robustness to Noise

In this experiment, two flows share a bottleneck bandwidth of 1 Mbps with an end to end RTT of 60ms. Uniform noise is generated with mean varying from 0 to 0.1 on the delay measurements. Two experiments were performed. In the first both the flows were prone to noise and in the second one, only one flow is prone. These experiments were repeated with FAST and SACK. With PERT, the Link drop rate remained zero throughout and the queue lengths were negligible. The link was almost fully utilized. Fig. 9 shows the link utilization for different protocols as the mean is varied, for both the experiments. This shows that for PERT noise doesn't impact the deduction of congestion and it maintains good Link utilization even at higher levels of noise.

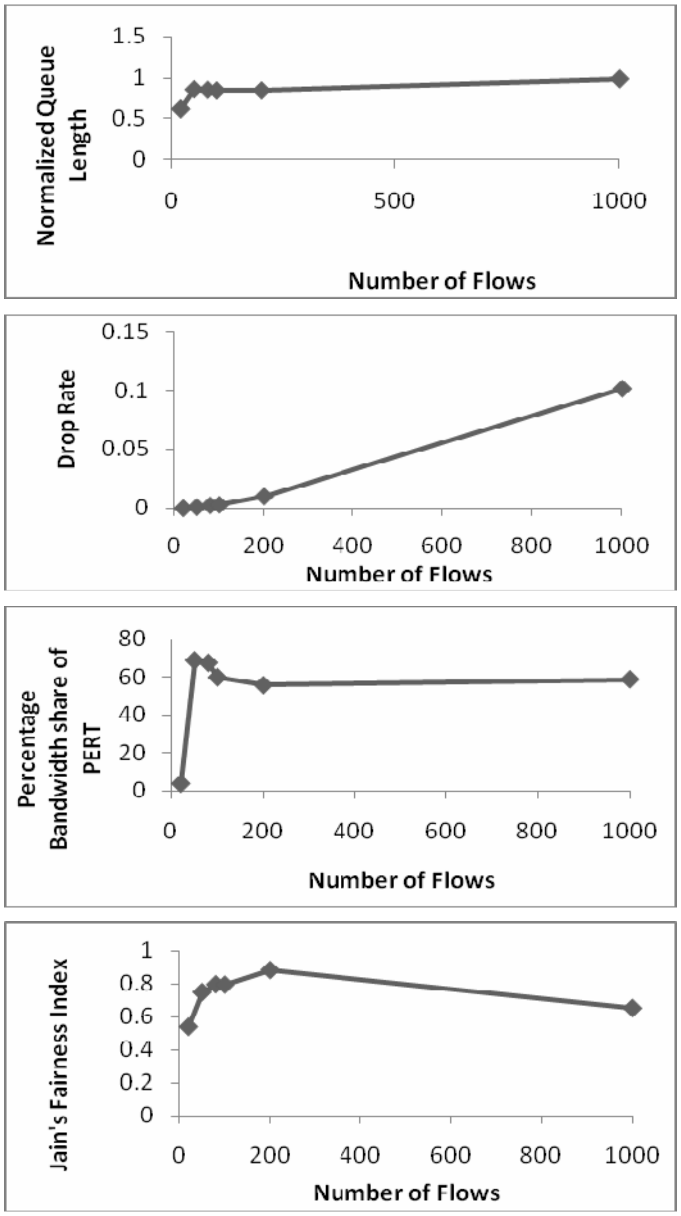
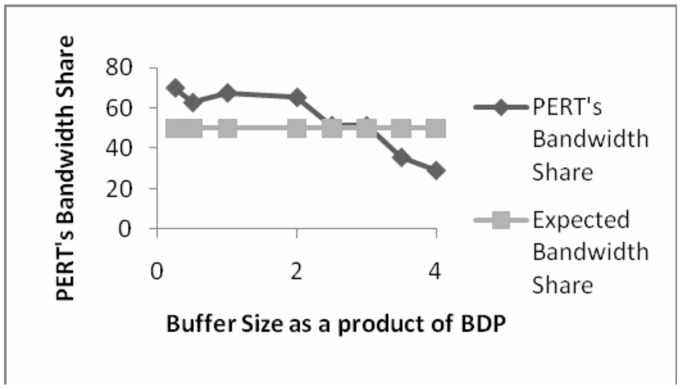


Figure 6: Variation of normalized queue length, drop rate, PERT's bandwidth share and Jain's fairness Index for a 50-50 scenario with number of flows.



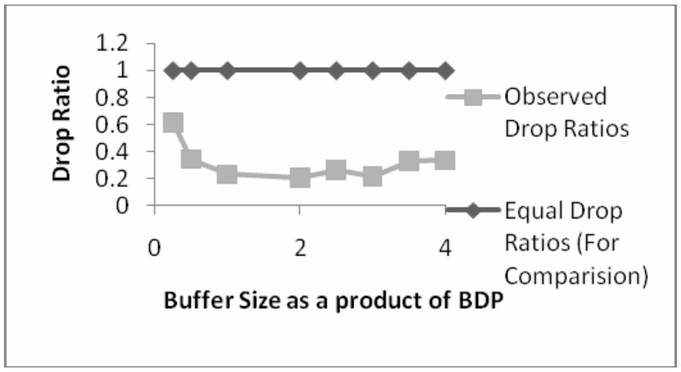


Figure 7: Variation of PERT's Bandwidth share and PERT to SACK drop ratio with Buffer size at the Router.

4.6 Tolerance to Channel Errors

PERT has an inherent robustness to channel errors. PERT's window reduction factor β depends on the queue length ($Srtt_{0.99}$) as mentioned earlier. When channel errors occur, if the queue length is low, PERT responds by reducing the window by a smaller factor on such errors thus resulting in higher throughput than that of other TCP flavors employing a window reduction factor of 0.5. Similar approach has been adopted in [18].

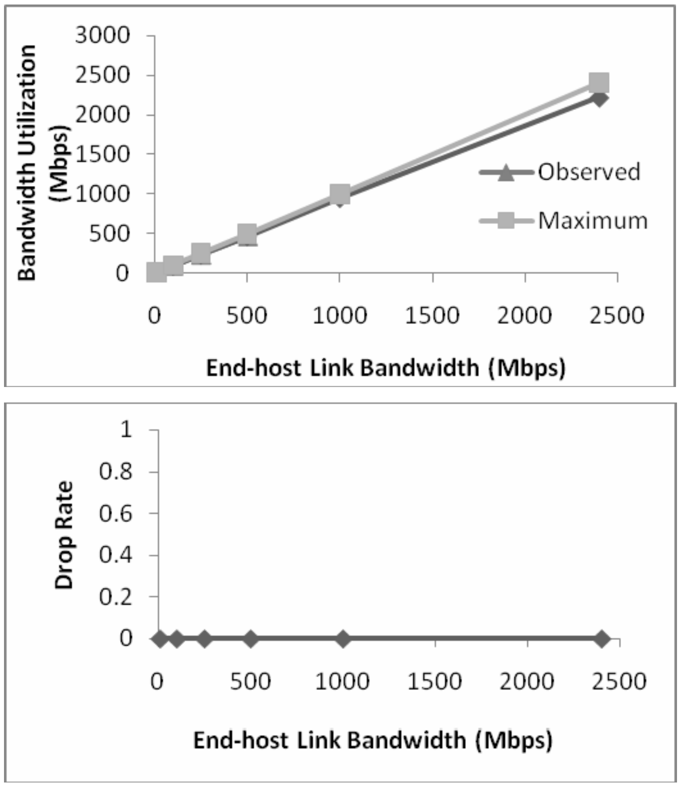


Figure 8: Variation of Utilization and drop rate at different end-host link bandwidths.

To evaluate the impact of channel errors on PERT, we considered two scenarios. In the first experiment, a single flow shares a bottleneck bandwidth of 1Gbps and the end to end RTT is 60ms. Random errors were induced using uniform loss model. Fig. 10 plots the throughput against the random loss

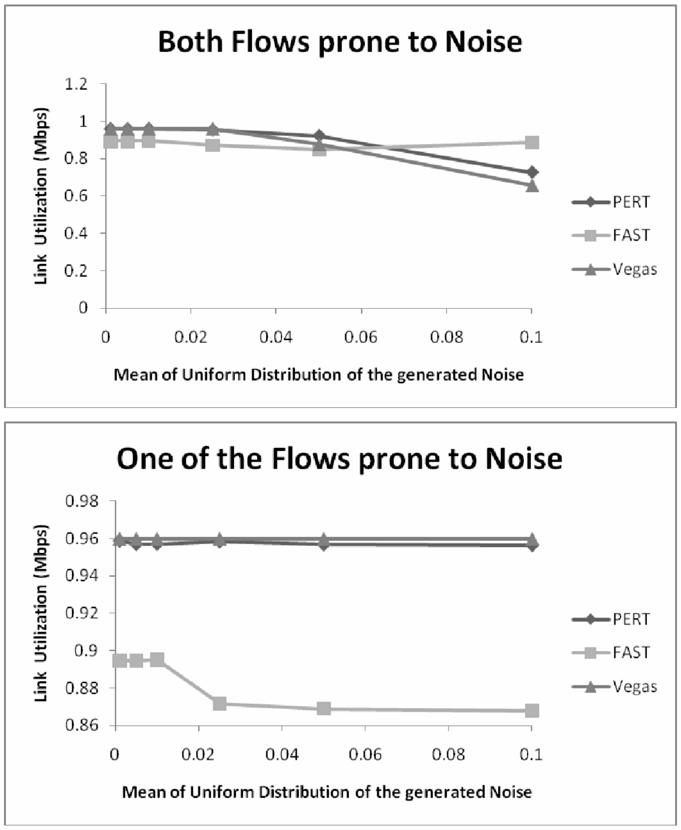


Figure 9: Variation of Link utilization with noise

rate. As the random loss rate increases, the link utilization of the SACK decreases drastically. PERT performs very well compared to SACK with higher utilizations until an error rate of 10^{-5}. The utilization deteriorates only at higher drop rates of order 10^{-4}. This shows that PERT tolerates channel errors gracefully in high-speed networks. We have also simulated an end-host wireless network of 55Mbps bandwidth and 20ms delay in a similar configuration with a source bandwidth of 100 Mbps and 5ms delay. However, in this case we varied the channel error rate till 10^{-3} and the results are shown in the second graph of Fig.10. As we can see, the utilization deteriorates only at much higher drop rates of order 10^{-3}. This shows that PERT performs well in normal wireless scenarios as well. This can be explained as follows. As channel errors are non-congestion errors, the queue may not be full when a channel error occurs. Moreover, PERT is designed to yield lower queue lengths when operating alone. Thus, we respond less on such a loss. This leads to a higher utilization compared to that of SACK, which always responds to a loss by a factor of 0.5.

4.7 Impact of Round Trip Delays, Number of Long-term flows and Web Traffic

We performed several experiments to see if the modified version (that can coexist with TCP) retains the properties of original PERT, when operating alone. In different experiments, RTTs and Number of long-term flows were varied. Results show that the properties of original PERT were retained.

Experiments were also conducted by introducing varying number of short-term web flows. The observed results closely matched the results of the original PERT.

We show some of those results here. Fig. 11 shows the results when the number of short-lived flows (web sessions) is varied from 10 to 1000, while keeping the long-lived flows constant. As seen from Fig. 11, as the load offered by the web traffic increases, the average link queue length remains low and as a result negligible packet losses are observed in case of PERT. On the other hand sack has higher queue lengths and higher drop rates.

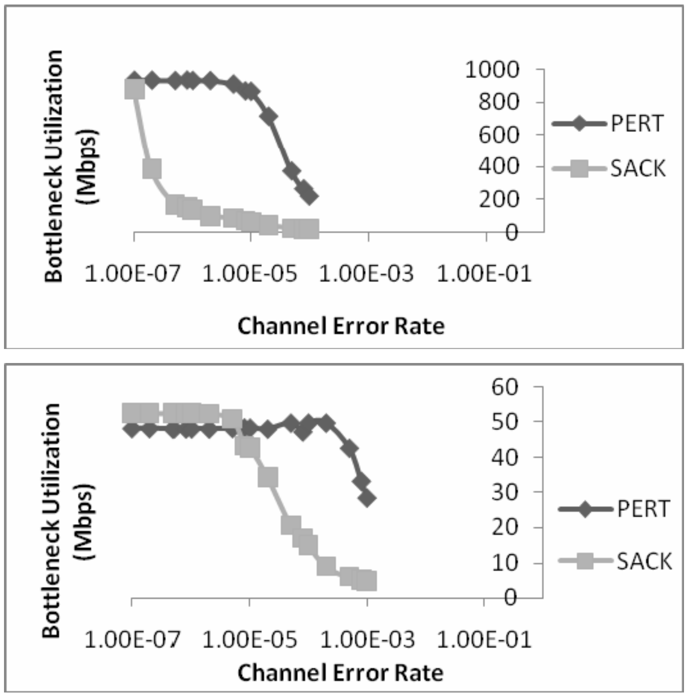


Figure 10: Variation of Bottleneck Utilization with channel error rate in high-speed and normal wireless networks.

In another experiment, the number of long-term PERT flows is varied from 1 to 1000, keeping other parameters constant. Fig. 12 shows the drop rate. Also Normalized queue length (not shown here) reaches at most 0.35 that is when there are 1000 flows sharing 500Mbps link. On the other hand the drop rate stays at zero irrespective of number of flows.

In another experiment, the end-to-end delay is varied in the range of 10ms to 1 second. Fig. 13 shows the drop rate. From the figure, we see that the drop rate is non-zero when the delay is low, as expected, and is zero for rest of the cases.

These results track the behavior of original PERT in homogenous deployment environment as reported earlier [3].

4.8 Results from live networks

We have implemented PERT in the network stack of the Linux 2.6.18 kernel. The network stack in the 2.6.x kernel is quite sophisticated and supports several standards from the RFCs as well as features beyond those published in RFCs or IETF Drafts aimed to provide good network performance [19].

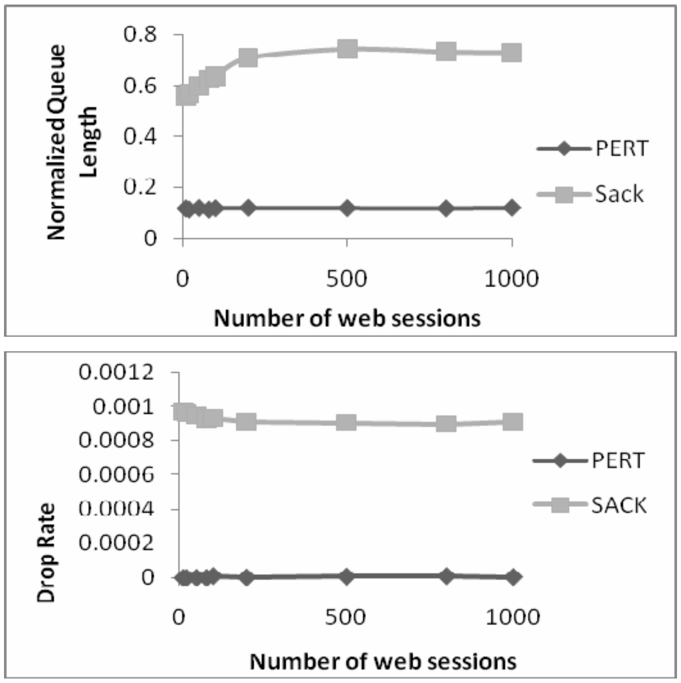


Figure 11: Variation of normalized queue length and drop rate with the number of web sessions.

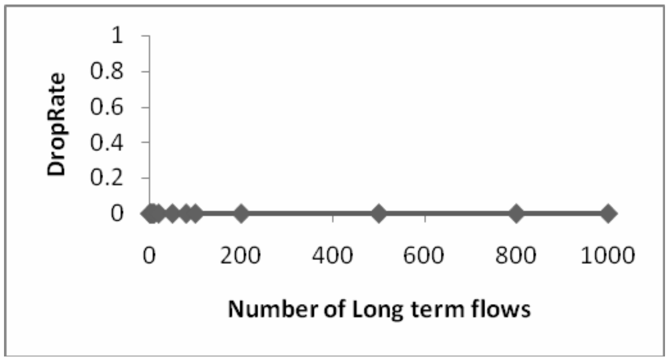


Figure 12: Variation of Drop rate with the number of long-term flows.

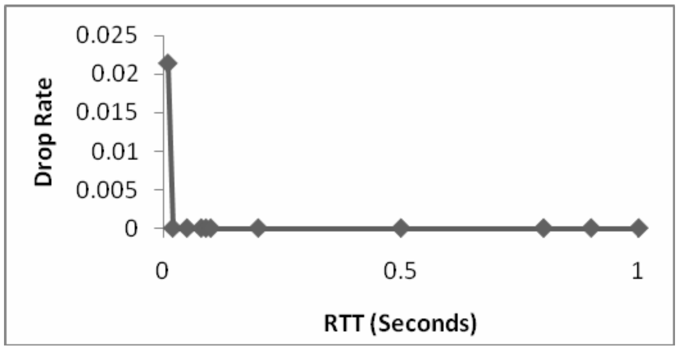


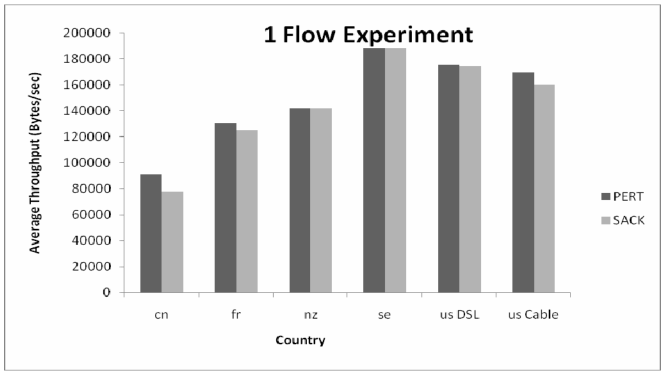
Figure 13: Variation of drop rate with RTT

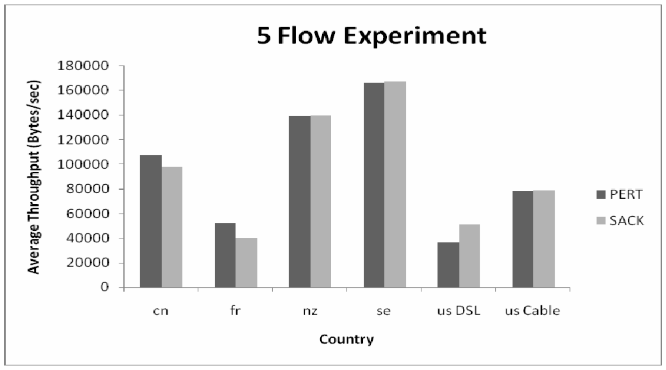
Our test bed consists of two off-the shelf Dell Optiplex GX260 workstations with Pentium 4 3.06GHz CPU, 1GB of RAM, Intel PRO/1000 MT gigabit NICs on to a 33MHz/32bit PCI bus. The two computers are connected to the Internet. One of them has the modified Linux kernel with PERT implementation and the other has the standard Linux kernel which uses SACK over New-Reno. To see how PERT works in the real-world traffic, we selected nodes on planet-lab across the globe, at six physically distant countries, to act as clients. We configured the two machines in our lab, which are in the same subnet, as servers. Data for throughput and losses were collected. Experiments were done with 1 flow, 2 flows and 10 flows of each flavor competing with corresponding number of flows of the other flavor. The experiments at each node are performed 10 times, to take into account the frequently varying traffic, and the results presented are averaged over different iterations. Fig. 14 shows the variation of throughput across different nodes in planet-lab. We observe that throughputs of both PERT and SACK are similar across different nodes, while PERT gets a little more than that of SACK in most cases.

The bar graphs in Fig. 15 show the variation of average number of drops at planet-lab nodes in different locations. The locations in order are China (cn), France(fr), New Zealand(nz), Sweden(se), US (DSL modem at College Station) and US(Cable modem at College Station) respectively. We see from the figure that number of drops for PERT is less than or equal to that of SACK in almost all the cases, despite sharing the core network links with numerous multiplexed flows, possibly using several different flavors of TCP. This shows that PERT performs well with the real-world traffic as well.

4.8.2 Results from Cable and DSL modem hosts

Recent measurements of RTTs using Cable and DSL modem hosts [17] show that there is a high variation in the RTTs measured from these hosts. We tested PERT by setting up these types of hosts as receivers and found its effectiveness. The throughput results were presented in section 4.8, where it was shown that PERT could compete with TCP in DSL and Cable access networks despite the RTT variability. In Fig. 16, we plot the instantaneous RTT and Smoothed RTT ($Srtt_{0.99}$) over time (jiffies) using the results from emulations, to show that $Srtt_{0.99}$ is a smooth signal despite highly varying instantaneous RTT signal. This shows that while instantaneous RTTs could vary significantly from sample to sample, the smoothed RTT employed by PERT as congestion signal is still effective in correctly gauging the congestion in the network. We have carried out extensive tests in these networks to test the effectiveness of PERT in both DSL and Cable networks and found that PERT is not affected by instantaneous RTT





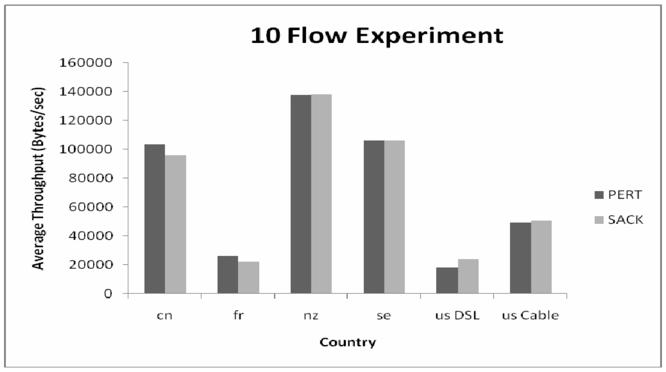
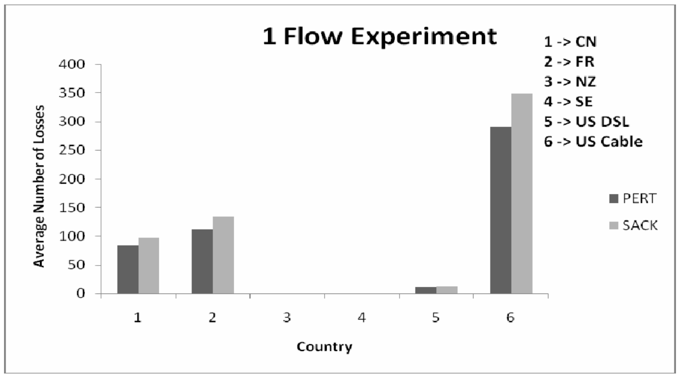
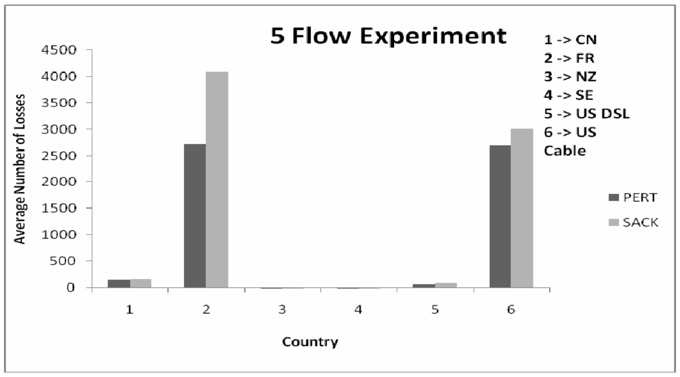


Figure 14: Variation of Throughput at different nodes across the world.

variability that is inherent in these networks due to the scheduling and access granting mechanisms employed in these networks.





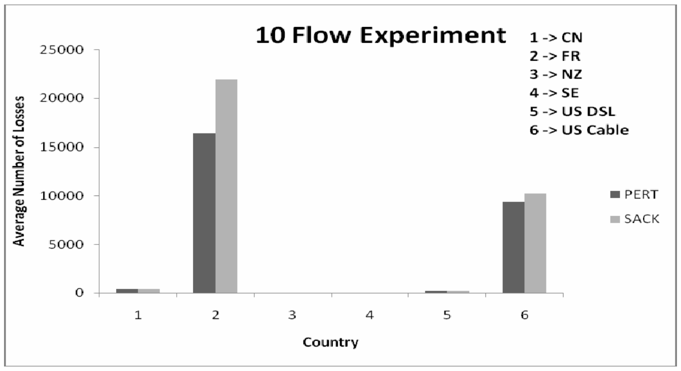


Figure 15: Number of Losses at different nodes across the world

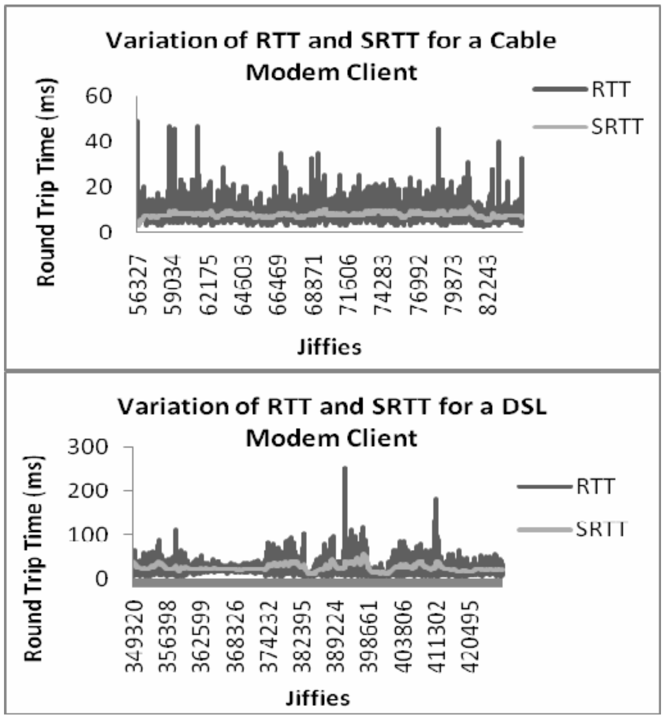


Figure 16: Variation of instantaneous RTT and Srtt$_{0.99}$ in Cable and DSL modem hosts

V. CONCLUSION

In this paper, we have presented rationale and design modifications for adapting the delay based protocol PERT to work well in heterogeneous environments, while preserving its original properties when operating alone. While, some earlier work has been done in making delay based protocols compete with loss based protocols, we are not aware of any work that simultaneously deals with incremental deployability and fair bandwidth sharing with currently dominant protocol TCP. We hope our work provides some insights into these issues of incremental deployability of new protocols and providing incentives (lower packet drop rates, in our case) for new protocol deployment. We have also tested PERT in real networks with different access networks including campus, DSL and Cable networks to observe that instantaneous RTT variability didn't impact the delay-based protocol's performance. We plan to investigate the issues in improving global network characteristics as a result of incremental deployability of new protocols such as PERT in the future.

ACKNOWLEDGMENTS

This work is supported in part by an NSF and Qatar National Research Fund. Discussions with Sumitha Bhandarkar have helped the paper.

REFERENCES

[1] Ravi S. Prasad, Manish Jain, Constantinos Dovrolis, "On the effectiveness of Delay-based Congestion Avoidance", *PFLDNet 2004*, February 2004.
[2] S. Rewaskar, J. Kaur and D. Smith, "Why Don't Delay-based Congestion Estimators Work in the Real-world?", Technical Report TR06-001, Department of Computer Science, UNC Chapel Hill, July 2005.
[3] S. Bhandarkar, A. L. N. Reddy, Y. Zhang, and D. Loguinov. Emulating AQM from end hosts. In Proc. ACM SIGCOMM'07.
[4] Bhandarkar, S., Jain, S., and Reddy, A. N. 2006. LTCP: improving the performance of TCP in highspeed networks. *SIGCOMM Comput. Commun. Rev.* 36, 1 (Jan. 2006).
[5] Cheng Jin, David X. Wei and Steven H. Low, "FAST TCP: motivation, architecture, algorithms, performance", *IEEE Infocom*, March 2004.
[6] Lisong Xu, Khaled Harfoush, and Injong Rhee, "Binary Increase Congestion Control for Fast Long-Distance Networks", To appear in *Proceedings of IEEE Infocom 2004*, March 2004.
[7] D.J. Leith and R. Shorten, "H-TCP Protocol for High-Speed Long Distance Networks", *PFLDNet 2004*, February 2004.
[8] Sally Floyd, "HighSpeed TCP for Large Congestion Windows", *RFC 3649* December 2003.
[9] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A self-configuring RED gateway", *IEEE INFOCOM'99*, March 1999.
[10] Bansal, D.; Balakrishnan, H., "Binomial congestion control algorithms," *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* , vol.2, no., pp.631-640 vol.2, 2001
[11] Deepak Bansal , Hari Balakrishnan , Sally Floyd , Scott Shenker, Dynamic behavior of slowly-responsive congestion control algorithms, Proceedings of the 2001 ACM SIGCOMM, p.263-274, August 2001, San Diego, California, United States.
[12] G. Appenzeller, I. Keslassy, and N. Mckeown, "Sizing router buffers", *ACM SIGCOMM'04*, August/September 2004.
[13] Y. Zhang, "Modeling and Stability of PERT", Texas A&M University Tech. Report, TAMU-ECE-2007-03, May 2007.
[14] Floyd, S., Handley, M., Padhye, J., and Widmer, J. 2000. Equation-based congestion control for unicast applications. In *Proceedings of ACM SIGCOMM,* Stockholm, Sweden, August 28 - September 01, 2000.
[15] Floyd, S. and Paxson, V. 2001. Difficulties in simulating the internet. *IEEE/ACM Trans. Netorkin,.* 9, 4 (Aug. 2001), 392-403.
[16] 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", *ACM SIGCOMM*, September 1999.
[17] Dischinger, M., Haeberlen, A., Gummadi, K. P., and Saroiu, S. 2007. Characterizing residential broadband networks. In *Proceedings of the 7th ACM SIGCOMM Conference on Iinternet Measurement* (San Diego, California, USA, October 24 - 26, 2007), 43-56.
[18] S. Liu, T. Basar and R. Srikant, "TCP-Illinois: A loss and delay-based congestion control algorithm for high-speed networks", Proc. of 1st Int. Conf. Perf. Evaluation Methodologies and Tools, Oct. 2006.
[19] P. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP", *Usenix 2002/Freenix Track, pp. 49–62*, June 2002.
[20] B. Wang, W. Wei, Z. Guo and D. Towsley, "Multipath live streaming via TCP: Scheme, Performance, Benefits", Proc. of CoNext, Dec. 2007.
[21] K. Tan, J. Song, Q. Zhang and M. Sridharan, "A Compound TCP approach for high-speed and long-distance networks", Proc. of IEEE Infocom, 2006.