ADAPTING A DELAY-BASED PROTOCOL TO HETEROGENEOUS ENVIRONMENTS

A Thesis

by

KIRAN KOTLA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2008

Major Subject: Computer Engineering

ADAPTING A DELAY-BASED PROTOCOL TO HETEROGENEOUS ENVIRONMENTS

A Thesis

by

KIRAN KOTLA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Co-Chairs of Committee,	A. L. Narasimha Reddy
	Riccardo Bettati
Committee Members,	Dmitri Loguinov
Head of Department,	Valerie E. Taylor

August 2008

Major Subject: Computer Engineering

ABSTRACT

Adapting a Delay-based Protocol to Heterogeneous Environments. (August 2008)

Kiran Kotla, B.E., Birla Institute of Technology and Science, Pilani, India

Co–Chairs of Advisory Committee: Dr A. L. Narasimha Reddy
Dr Riccardo Bettati

We investigate the issues in making a delay-based protocol adaptive to heterogeneous environments. We assess and address the problems a delay-based protocol faces when competing with a loss-based protocol such as TCP. We investigate if noise and variability in delay measurements in environments such as cable and ADSL access networks impact the delay-based 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 the TCP flavor SACK. We show through simulations and real network experiments that, with the proposed changes, PERT experiences lower drop rates than SACK and leads to lower overall drop rates with different mixes of PERT and SACK protocols. Delay-based protocols, being less aggressive, have problems in fully utilizing a highspeed link while operating alone. We show that a single PERT flow can fully utilize a high-speed, high-delay link.

We performed several experiments with diverse parameters and simulated numerous scenarios using *ns-2*. The results from simulations indicate that PERT can adapt to heterogeneous networks and can operate well in an environment of heterogeneous protocols and other miscellaneous scenarios like wireless networks (in the presence of

channel errors). We also show that proposed changes retain the desirable properties of PERT such as low loss rates and fairness when operating alone.

To see how the protocol performs with the real-world traffic, 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 at diverse location across the globe on the planet-lab.

The results from simulations indicate that PERT can compete with TCP in diverse environments and provides benefits as it is incrementally deployed. Results from real-network experiments strengthen this claim as PERT shows similar behavior with the real-world traffic.

To my parents

ACKNOWLEDGMENTS

I thank my research advisor, Dr. A. L. Narasimha Reddy, for giving me this valuable chance to work under him. As a mentor and critic, he has helped me not only with my research but also to motivate and challenge myself and to push my limits. He always made sure that I did not stray too far away from the main objective of my research. I specially thank him for all this, thank you very much, Dr. Reddy. I thank Sumitha Bhandarkar, an earlier student of Dr.Reddy, for her constant assistance in terms of providing valuable help at times despite her busy schedule as a full-time researcher at Motorola. I thank the staff and system administrators in the Computer Engineering group of the ECE department at Texas A&M University, for their timely help for procuring machines and software support. The other students in my research group have been very helpful to me at times and I would like to express my kind gratitude to them. Finally, I thank my parents - who have always supported me at hard times and for encouraging my love for learning and the encouragement to pursue my master's. Last, but not the least, I thank the Almighty God for giving me strength to overcome my limitations all the time.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

The dominant transport layer protocol currently and for the past two decades in the Internet has been TCP. TCP uses Additive Increase, Multiplicative Decrease window adjustment mechanism by which it adapts to diverse scenarios. The congestion control algorithms associated with the standard TCP, like slow-start and congestion avoidance, have been crucial to ensuring the stability of the Internet.

Congestion Protocols can be broadly categorized into two types namely congestion avoidance schemes and congestion control schemes. The TCP Additive Increase Multiplicative Decrease (AIMD) scheme coupled with Fast retransmit/recovery is commonly referred as congestion avoidance mechanism. However, the popular and widely used TCP congestion flavors like TCP-Newreno and TCP-SACK, detect congestion only when a packet loss occurs and hence are congestion control schemes. A typical congestion avoidance schemes predicts congestion at the router at an early stage by actively monitoring either the RTTs of its packets or the throughput. Based on its prediction, it judges whether to reduce its window to avoid possible packet losses. Therefore, congestion avoidance protocols respond early to the congestion.

There are two different schools of thought concerning congestion avoidance. The first being congestion avoidance at routers and the second being at the end-hosts. As routers typically operate at a junction of different sources sending packets to different destinations, they will be in a better position to judge when the congestion actu-

---

The journal model is *IEEE Transactions on Automatic Control*.

ally builds up. Therefore, they can prevent congestion by carefully judging to drop packets at an earlier stage (implicitly signaling congestion) or explicitly informing the end-host about the incipient congestion through markings on packets. Such mechanisms are commonly referred to be a part of Active Queue Management (AQM). [1] and [2] are examples of AQM mechanisms.

While routers are at a better position to determine the onset of congestion, it is often difficult to deploy the router based mechanisms, as it requires changes not only at the routers but also at the end-hosts. Therefore, the other school of thought assumes that the network to be a black-box and employs its own mechanisms to detect and avoid congestion. Over time many congestion avoidance protocols like TCP-Vegas [3] were proposed. However, such protocols are inherently vulnerable to noise and sudden changes in the delay measurement. Furthermore, several challenges in estimating delays accurately and other issues have resulted in skepticism in the viability of delay-based schemes [4], [5].

Recently, Sumitha et al. identified the problem of noise vulnerability of delay-based protocols and addressed some of these issues and proposed a delay-based protocol congestion avoidance protocol named PERT (Probabilistic Early Response TCP) [6]. PERT improves the delay estimation process by maintaining an Exponentially Weighted Mean Average (EWMA) of the measured RTTs and deals with remaining uncertainties through a probabilistic response to the congestion identified by the estimated delay. PERT emulates the behavior of AQM at end hosts by responding at a higher rate at higher delays and at a lower rate at lower delays. While PERT has been shown to be effective in reaching its goals, a number of technical challenges remain open in its practical deployment in the real-world.

The major issue is how delay-based protocols can compete with various versions of loss-based protocols like TCP. Loss-based protocols keep increasing their rate until a packet is dropped (as most versions of TCP do). On the other hand, delay-based congestion avoidance protocols, by responding to congestion early, cede ground to such loss-based protocols. While most delay-based protocols exhibit good properties in a homogenous deployment, for the delay-based protocols to be deployed in the real-world, they need to be able to operate in an environment of mixed protocol deployment. This would be necessary for incremental protocol deployment.

PERT employs a good congestion prediction signal $srtt_{0.99}$ and estimates the queuing delay accurately enough to perform well in terms of queuing delays, drop rates and intra protocol fairness in diverse scenarios. Given these nice benefits, we performed several experiments to see how PERT behaves when it competes with loss-based protocols like standard TCP. We found out through our experiments that, like many proposed delay-based schemes, it loses to loss-based protocol like TCP and gets less than 1% bandwidth share in a 50-50 mix of PERT and TCP flows.

As mentioned earlier the current dominant protocol is TCP and if a proposed scheme does not get a reasonable bandwidth share when used in practice with other co-existing TCP flows, it will not be adopted. It is unreasonable and impractical to ask everyone to switch to delay-based protocols, at once, for better performance benefits. Thus, any protocol is generally of practical interest only if it can coexist with flows of dominant flavors in the Internet. Thus the major motivation to adapt PERT to heterogeneous environments is to make it compete for a fair bandwidth share when PERT has to co-exist with TCP in current Internet.

PERT's ability to compete with TCP, after modifications,should not compromise its attractive properties for adoption *i.e.,* low loss rates and low queue lengths for individual adopters of PERT. Low Queueu lengths cannot be guaranteed in a mixed deployment as queue lengths depend on the behavior of both PERT and TCP. Our next aim was to provide global network wide benefits or incentives for PERT's deployment. As more people adopt PERT, we want to see if we could provide lower total packet drops in the network in similar operating conditions.

We address this issue in this research. We propose and evaluate design enhancements to enable the delay-based protocol, PERT to compete with flavors of TCP like 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 such experiments that incremental deployment of PERT provides lower overall drop rates along with lower drop rates for the PERT flows in the mix. While some recent protocols [7–9] 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.

Another 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 [6]. Further, recent work on cable and ADSL access networks has highlighted the RTT variance of these networks even in the absence of congestion [10], 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 the modified PERT in cable and ADSL networks. We report on PERT's ability to correctly gauge congestion even 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. We show through these experiments that PERT can be more robust to noise in delay compared to other delay-based protocols like FAST and Vegas.

A third issue we address is whether a delay-based protocol can be scaled to provide high utilizations when a single flow operates in high-speed, high-delay links. This has been a topic of considerable interest lately and many new protocols have been proposed [7, 11–14]. We show that PERT can fully utilize high-speed, high-delay network links and provide near zero drop rates than loss-based schemes such as [11–13].

While FAST [7], 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.

We make the following significant contributions: (a) Adapted a delay-based pro-

tocol to compete with the existing loss-based congestion protocol TCP, (b) Extensive evaluations of mixed deployment scenarios of PERT and TCP to show that PERT offers benefits to individual adopters while providing overall incremental benefits to network characteristics and hence providing a path to incremental deployment. (c) Adapted the delay-based protocol 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).

CHAPTER II

A BRIEF DESCRIPTION OF PERT

Delay-based congestion control algorithms like TCP-Vegas were proposed to avoid packet losses and keep the queue lengths at the routers low. However, delay-based protocols can be vulnerable to the noise and sudden variations in the RTT measurements. The major challenge for any delay-based protocol is to carefully judge the congestion that is building up at the router. In an end host based congestion control, the congestion can be classified into three different states namely low-congestion (*i.e.,* when low delays are observed), high-congestion state (*i.e.,* when higher delays are observed and queue starts getting filled up) and the packet-loss state when the queue is full and a packet loss occurs. If a delay-based flow does not use an accurate method for determining congestion, or if the flow shares the bottleneck link with a lot of cross traffic and flows of other flavor, then the duration of high-congestion state may bee to short to be able to detect it. This is known as a "false negative". On the other hand, a protocol can be too aggressive or unreliable in predicting congestion, and as a result may unnecessarily reduce its sending rate and face performance degradation. This is referred to as "false positive."

When different protocols were compared in terms of their prediction efficiency *i.e.,* low false positive and low false negative rates, Vegas [3] was found to have the best prediction efficiency [6]. These results indicated that there was room to omprove on the congestion prediction schemes employed at that time. Such experimental results guided the design of a better congestion predictor employed in PERT [6].

PERT has shown that end-host based congestion prediction can be more accu-

rate than previously studied. However, it is still not possible to completely eliminate the uncertainty in the congestion prediction using RTT measurements. To address these problems a probabilistic early response mechanism was used in PERT. PERT emulates the behavior of AQM/ECN at the end hosts. The benefits were shown to be similar to that of using an AQM mechanism at the router and better than the existing delay-based schemes such as TCP-Vegas, without requiring any form of support from the router.

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 [1]. 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, PERT defines 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 11. For the parameters *minthresh_*, *maxthresh_* and *maxP_* PERT uses 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 [15].

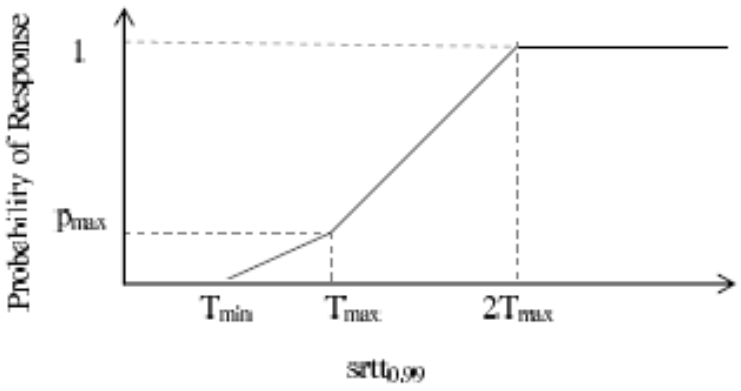

Fig. 1. Probabilistic response curve used by PERT

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

When operating in a homogeneous environment, PERT has been shown to utilize links fully while providing delays and nearly zero packet loss rates. We propose modifications to PERT to make it incrementally deployable in a network that is TCP

dominated. We show through our evaluations that with the proposed modifications PERT retains many of the original properties of low loss rates and queue lengths.

CHAPTER III

PROPOSED APPROACH

As PERT is a delay-based approach, it responds relatively earlier than loss-based schemes to congestion. PERT, by responding early to the congestion early, cedes ground to loss-based schemes in claiming bandwidth. As we do not intend to change this early response mechanism, one way that remains open to make PERT more aggressive is to change the parameters α and β employed in the window adjustment schemes $W = W + \alpha/W$ on an *ack* in the congestion avoidance phase and $W = (1 - \beta) * W$ on a congestion response.

A. Throughput analysis

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. The steady state throughput of a flow that follows the window adjustment schemes mentioned above is given by $\frac{1}{RTT} * \sqrt{\frac{\alpha}{\beta * p}}$ [16]. 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 [17], but for simplicity of analysis, we assume that SACK and PERT observe similar congestion loss probability.

1. α adjustment

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 state throughput equations of the two protocols [17], 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 PERT $W = W + \alpha/W$, where $\alpha_{PERT} = 1 + p'/p$.

2. β adjustment

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 [18], 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 [18] 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.

B. Detailed description

The local stability of PERT under these changes has been analyzed [19]. The analysis shows that these changes actually improve the local stability region of PERT [19]. We omit the details here and refer you to [19] for the details of analysis.

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, here), it deciphers that the bandwidth is being under utilized and increments α linearly till it reaches a threshold of 32, to increase its window during the congestion avoidance phase. We call this High-speed region or mode. 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. We call this safe or moderate mode. 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 $\alpha_{PERT} = 1 + p'/p$. This adaptation of α takes place at slightly longer time scales than an RTT (we used *5*RTT* in our implementation). We call this compete mode. The target alpha is

smoothly varied as the buffer transitions from one region to another region. Fig. 2 shows the variation of alpha with the estimated delay ($srtt_{0.99}$).

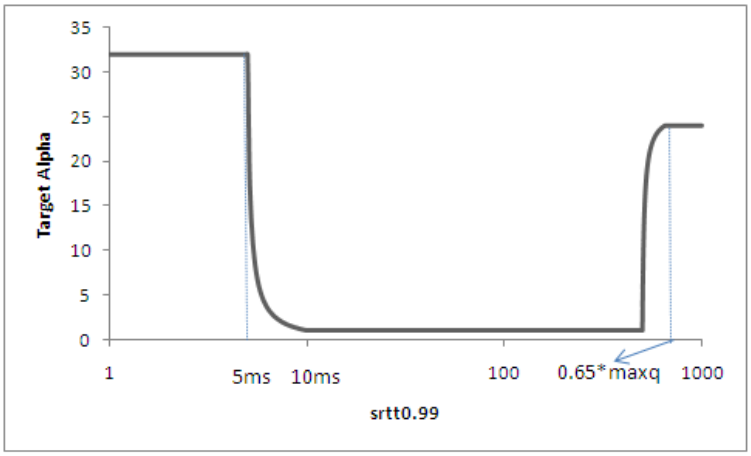

Fig. 2. Variation of target alpha with the estimated delay

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. Larger number of packet drops results in degraded performance in terms of throughput. Increased queue lengths result in larger queuing delays. To counter these problems, we set $\alpha_{PERT} = min(\alpha_{max}, 1+p'/p)$ where α_{max} is a parameter chosen to control this burstiness. In our simulations and emulations below, we chose α_{max} to equal 32. It seems feasible to make this parameter dependent on both the observed drop rate and observed queuing delays. However, such modifications will result in less intra protocol fairness as different flows may see different RTTs and a even a slight difference in the estimation of RTT between flows results in a significant difference of parameters

resulting in degraded fairness among flows. We will explore this problem in the future.

CHAPTER IV

IMPLEMENTATION ISSUES AND PARAMETER TUNING

We implemented the above modifications to PERT in *ns-2* and linux kernel 2.6.18. The major challenges we faced during the implementation of the modified PERT were to estimate the packet drop rate p and early response rate p'. We tried several implementations ranging from simple schemes like 1/N, where 'N' is the number of packets between two consecutive drops for estimating drop rate and two consecutive early responses for estimating early response rate, to other complex schemes. Though they were very simple to implement, they were not accurate enough for the scheme to work well and did not yield consistent results for different experiments that we conducted.

After a careful study of the existing mechanisms for estimating drop rates, we came across the TFRC algorithm for estimating the packet drop rate p [20]. We used a similar mechanism for our drop rate estimation and this yielded better and consistent results. The TFRC packet drop estimation algorithm maintains details of window of the last eight packet losses in the form of a window number of packets that were successfully transmitted between two consecutive losses. All these eight values are taken into account while estimating the overall packet loss, by giving more weights to the recent details and lesser weights to the older details. To be precise, the recent four losses are given a weight of '1' each and the last four are given weights 0.8, 0.6, 0.4 and 0.2 respectively. The overall packet loss is estimated as a mean of all these values. Therefore, $lossrate = \frac{6}{(n1+n2+n3+n4+0.8*n5+0.6*n6+0.4*n7+0.2*n8)}$, which reciprocal of the mean number of packets transferred between successive losses. We initially set the early response rate p', to the actual probability of early response as

computed by the original PERT based on the estimated queuing delay. This did not yield results as good as expected. We therefore, employed a similar method to drop rate estimation mechanism, for estimating the early response rate of PERT *p'*.

Though the above scheme works well for estimating drop rate and early response rate which are crucial for the operation of the modified PERT, the values get obsolete when the number of drops or early responses are less and a large time elapses since the last drop or early response. To counter this problem, we periodically update the values of p and *p'* depending on the how obsolete the values are. To be precise, when the number of packets transmitted since last loss exceeds the mean number of packets (computed above), then the window is moved right for the first time, in the sense the oldest entry is flushed and the current number of packets since the last loss is set as the first window entry. This value keeps on getting updated till the next drop occurs, whenever the number of packets since last loss exceeds the mean number of packets computed as above from the window of last eight entries. This gives us better estimates of those values and yields consistent performance of the protocol.

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 gauge correctly the mode of PERT's operation (Safe vs Compete vs high-speed. 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.

Though the implementation of PERT is very straightforward in *ns-2*, this is not

so regarding the implementation in the Linux Kernel. This is because at the kernel level, there is no support for floating point arithmetic and we have to implement the arithmetic on our own as our scheme has a few variables that need to be as accurate as possible. We tried to be as accurate as possible at the Kernel level regarding the estimation of different parameters by implementing arithmetic to provide precision up to 6 decimals.

CHAPTER V

EXPERIMENTAL EVALUATION IN HETEROGENEOUS PROTOCOL ENVIRONMENTS

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. For the live network experiments, we make our evaluation realistic by choosing nodes at different countries to act as clients to download files for longer duration from servers at our lab.

A. Simulations

For all the simulation experiments, the bottleneck buffer size is set to the bandwidth-delay product, unless otherwise stated, 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. All the simulations follow the topology in Fig. 3, unless otherwise mentioned.

We consider several metrics for evaluation, bandwidth share of different protocols, drop rates and queue lengths at routers. In order to compare different protocols, we also compute "drop ratios". Drop ratio of a protocol is defined as the drop rate observed of that protocol divided by the drop rate of competing TCP flows. We also study the Jain's Fairness Index.

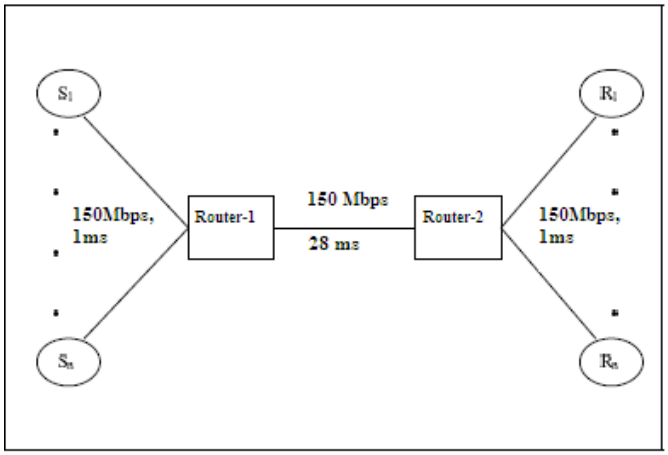

Fig. 3. Topology used for *ns-2* simulations

1. Varying mix

In this experiment, we vary the percentage of PERT flows in a mix of PERT and TCP flows. This experiment is conducted to observe the impact on network and protocol characteristics as PERT's deployment goes from 0% to 100%. 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. 4 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 as defined earlier, drop ratio is ratio of drop rates of PERT and SACK

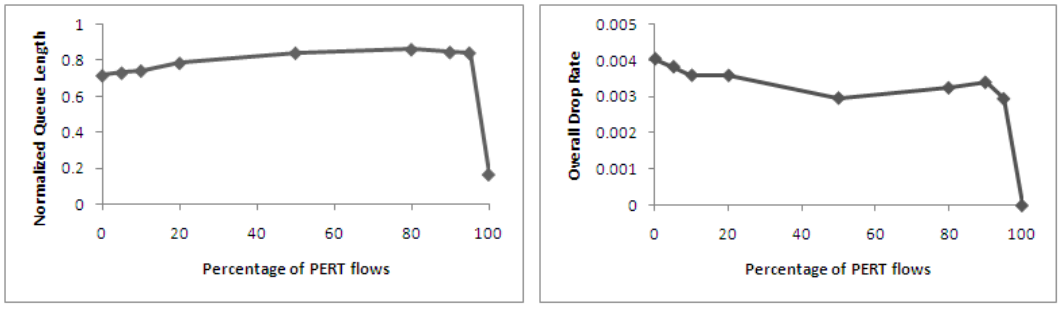

Fig. 4. Variation of drop rate and normalized queue length with percentage of PERT flows

and to be computed, flows of both the flavors need to be competing in the experiment. Fig. 5 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. 5 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.

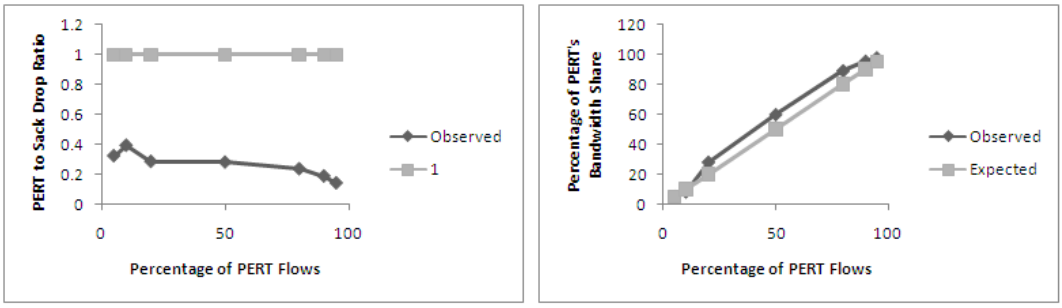

Fig. 5. Variation of PERT's bandwidth share and FAST to SACK drop ratio with percentage of PERT flows

Results in Fig. 4 and Fig. 5 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 here that bandwidth incentives are easy to provide compared to the gains observed here in packet loss rates. These results show that PERTs 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 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.

In the same experiment, we calculated the mean and variance of the bandwidth (measured once in a second) for PERT and SACK flows. Fig. 6 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.

An experiment with a similar setup is repeated for a mix of FAST [7]and SACK. Fig. 7 summarizes the results. From Fig. 7, 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 much larger share of bandwidth compared to SACK. These results indicate that FAST's modifications for competing against TCP may not be

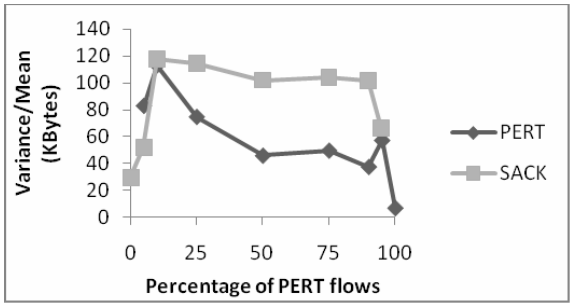

Fig. 6. Variation of the ratio variance/mean of PERTS bandwidth with percentage of PERT flows

beneficial to both TCP and itself as it results in higher drop rates, even in a 100% FAST environment. We explore the performance with FAST and other protocols in mixed scenarios in later sections.

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. 8, 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. 8 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 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 Jains Fairness index varies accordingly as PERT's bandwidth share.

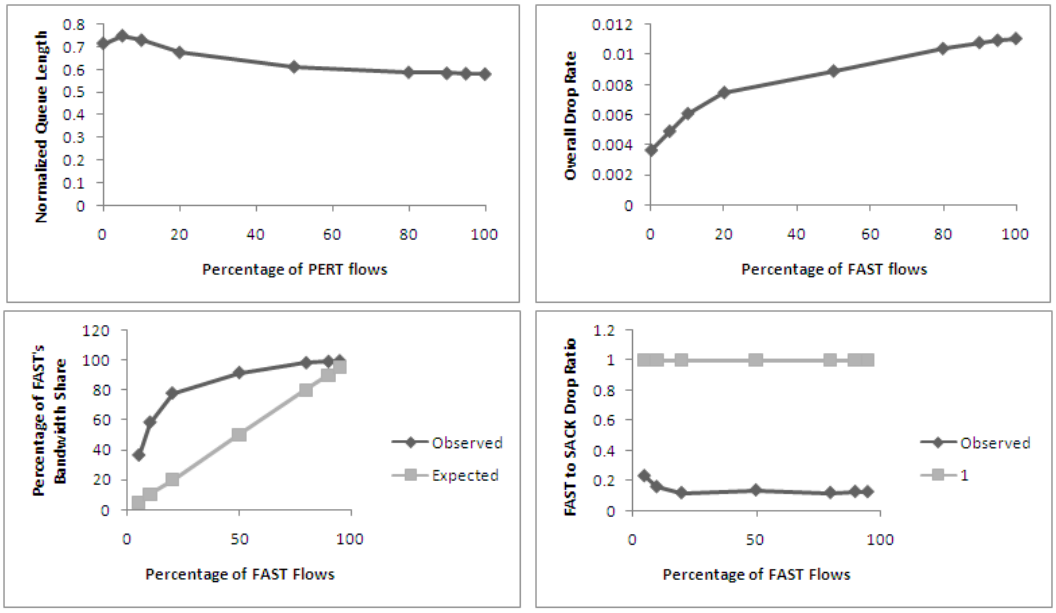

Fig. 7. Variation of normalized queue length, overall drop rate, FAST'S bandwidth share and FAST to SACK drop ratio with percentage of PERT flows

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. 9 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

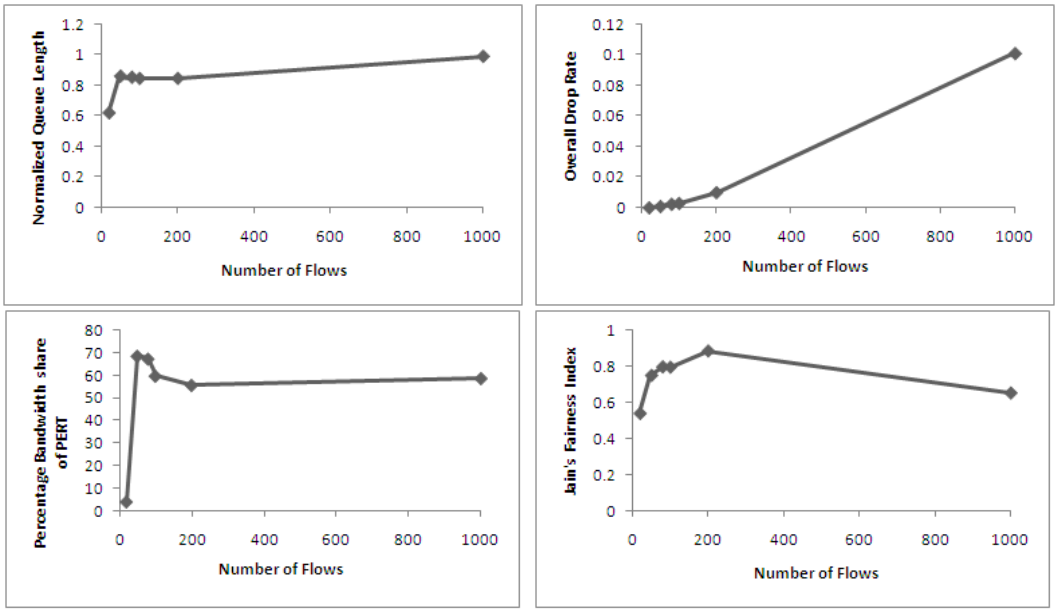

Fig. 8. Variation of normalized queue length, drop rate, PERTs bandwidth share and Jains Fairness Index for a 50-50 scenario with number of flows

loses to SACK in the presence of larger buffers (*3.5xBDP* 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*.

B. Co-existence of Illinois with PERT

An experiment with a similar setup as above is repeated for a mix of PERT and TCP-Illinois [8]. Fig. 10 summarizes the results. From Fig. 10, we see though the overall drop rate decreases with the increase in the percentage of Illinois flows, the drop rate at 100% PERT is far less than that of 100% Illinois. Moreover, the queue length increases with the increase in Illinois share, which implies that the queuing delay gets worse. Further, the drop ratio of Illinois to PERT is larger than 1 in most

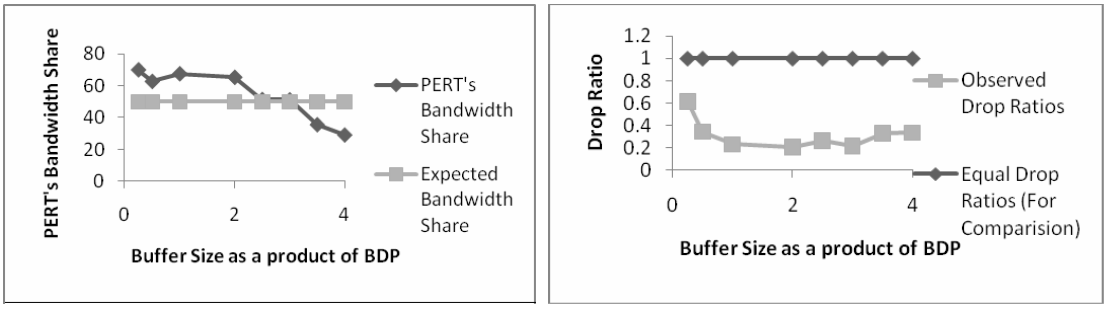

Fig. 9. Variation of PERTs bandwidth share and PERT to SACK drop ratio with buffer size at the router

of the cases implying that PERT will see less number of packet drops compared to Illinois. Though Illinois observes less bandwidth than expected, its parameters are easy to be tuned to increase the share. These results indicate that PERT can co-exist with TCP-Illinois.

C. Variation of number of flows in a 50-50 mix of PERT and Illinois

The experimental setup is similar to the above experiment, with 50% of PERT and 50% of Illinois flows in each case. Fig. 11 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. 11 that the PERT to Illinois drop ratio is low, implying that PERT sees lower drop rates in the mix over a wide range of number of long term flows. Further the bandwidth share of PERT always is also more than Illinois share. We can also see that the queue lengths are very high as is the case when flows of Illinois flavor are in the mix.

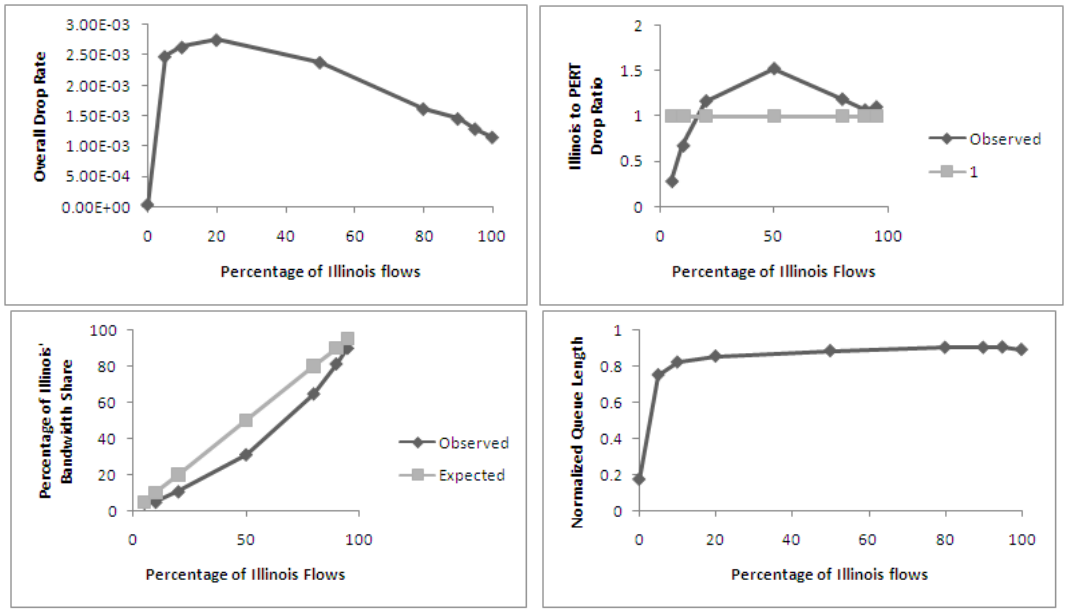

Fig. 10. Variation of normalized queue length, overall drop rate, Illinois bandwidth share and Illinois to PERT drop ratio with percentage of Illinois flows

D. PERT with non-responsive traffic

We wanted to see how PERT utilizes the link when there is intermittent non-responsive traffic like UDP. The experimental setup has a 1Gbps bottleneck and one PERT flow with RTT 60 ms tries to utilize the link. The duration of the simulation is about 900 seconds. The experiment starts with PERT alone utilizing the link and later once in every 150 seconds, one UDP flow of Constant Bit Rate (CBR) shares the link with PERT for a duration of 150 seconds. Fig. 12 shows the PERT's bandwidth share plotted at continuous intervals of time. We can see from the graph that PERT quickly adjusts its bandwidth share accordingly. In the presence of non-responsive traffic PERT reduces its sending rate. When the non-responsive flows are not present, PERT quickly ramps up its sending rate. This shows that PERT can react to network dynamics quickly.

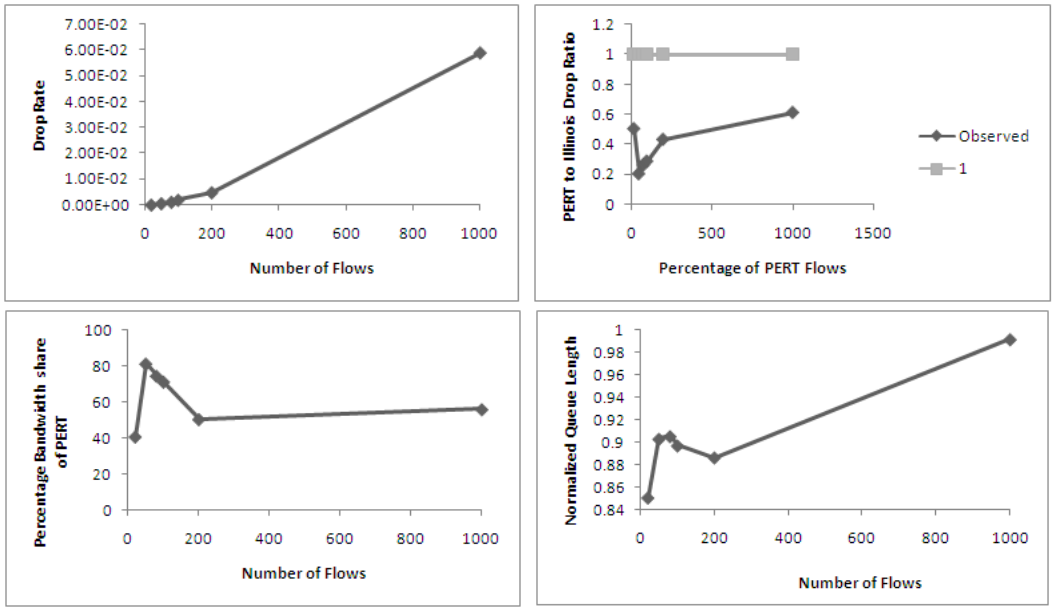

Fig. 11. Variation of drop rate, PERT to Illinois drop ratio, PERT's bandwidth share, normalized queue length,for a 50-50 mix of PERT and Illinois with number of flows

E. Live network experiments

Though some protocols claim to perform well using *ns-2* simulations, they often fail in practice. To make sure this is not the case with PERT, we implemented it in the network stack of the Linux 2.6.18 kernel and performed real-world emulations by choosing six different clients at geographically distant locations. To reflect changes in the diverse real-world traffic, experiments with each node were performed 10 times and averaged. We chose to work on the network stack in the 2.6.x kernel as it is quite sophisticated and supports several standards from the *RFC*s as well as features beyond those published in *RFC*s or *IETF* Drafts aimed to provide good network performance [21].

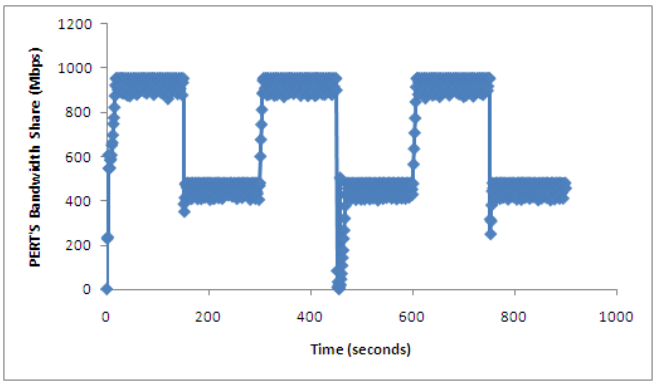

Fig. 12. PERT's bandwidth share with intermittent UDP traffic

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. As mentioned earlier, the experiments at each node were performed 10 times, to take into account the frequently varying traffic, and the results presented are averaged over different iterations. A 95% confidence interval is also shown for the results. Fig. 13 shows the variation of throughput across different nodes in planet-lab. We observe that throughput of both PERT and SACK are similar across different nodes, while PERT gets a little more than that of SACK in most cases.

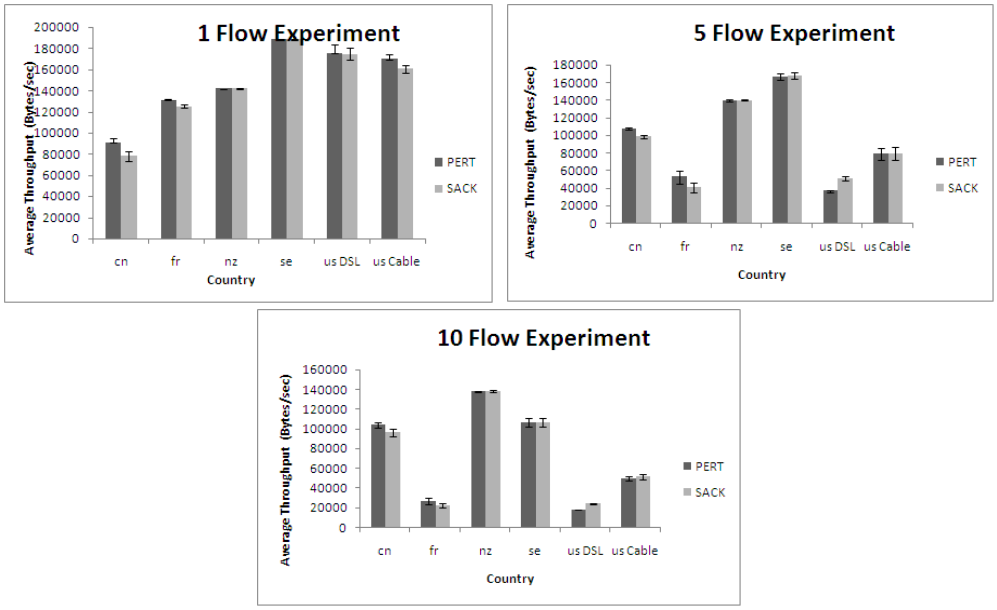

Fig. 13. Variation of throughput at different nodes across the world

The bar graphs in Fig. 14 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 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.

1. Results from cable and DSL modem hosts

Recent measurements of RTTs using Cable and DSL modem hosts [10] show that there is a high variation in the RTTs measured from these hosts. We tested PERT by

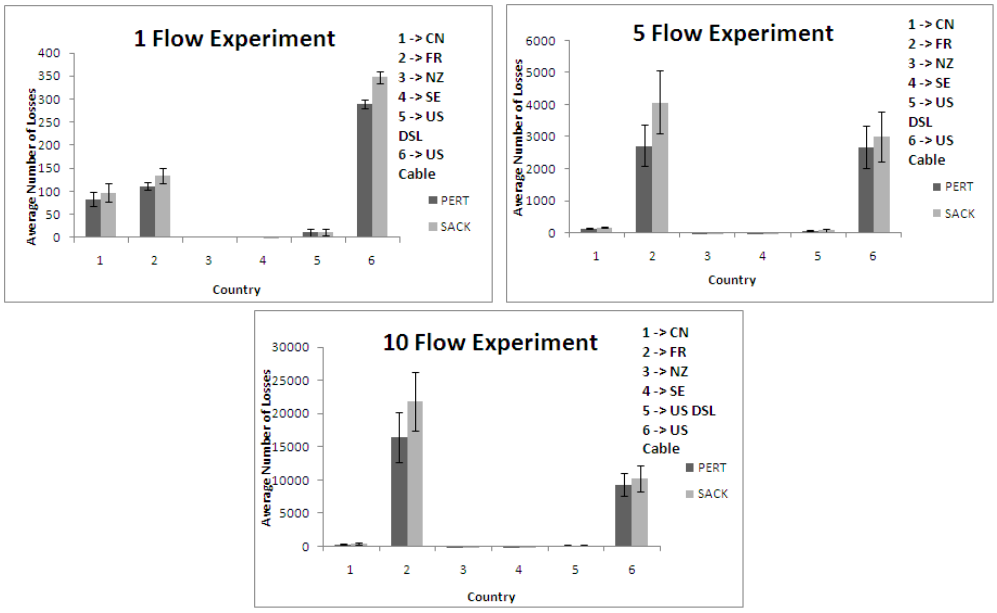

Fig. 14. Number of losses at different nodes across the world

setting up these types of hosts as receivers and found its effectiveness. The throughput results were presented above, where it was shown that PERT could compete with TCP in DSL and Cable access networks despite the RTT variability. In Fig. 15, we plot the instantaneous RTT and Smoothed RTT ($srtt_{0.99}$) over time (*jiffies*) using the results from network experiments, 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 variability that is inherent in these networks due to the scheduling and access granting mechanisms employed in these networks.

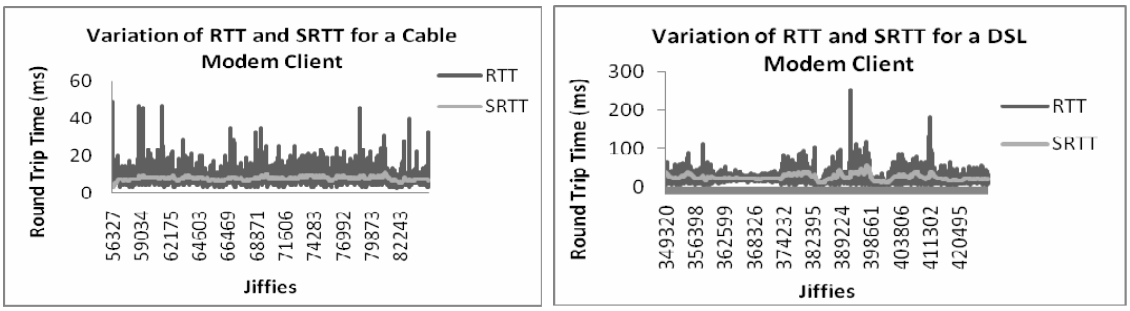

Fig. 15. Variation of instantaneous RTT and $srtt_{0.99}$ in cable and DSL modem hosts

2. Coexistence with CTCP

Microsoft's Compound TCP (CTCP) [9] has been deployed with Windows vista platform. We performed similar live experiments with one of our servers running PERT and the other running CTCP. We used the linux kernel implementation of CTCP, distributed for research purposes, for our experiments. Similar to the experiments with SACK, we perform experiments with 1, 5 and 10 flows of PERT and CTCP competing with each other. The experiments were performed 10 times and the results provided are the values averaged over different iterations. A 95% confidence interval is also shown.

Fig. 16 shows the variation of throughput across different nodes in planet-lab. We observe that throughput of both PERT and CTCP is comparable across different nodes.

Similarly, the bar graphs in Fig. 17 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

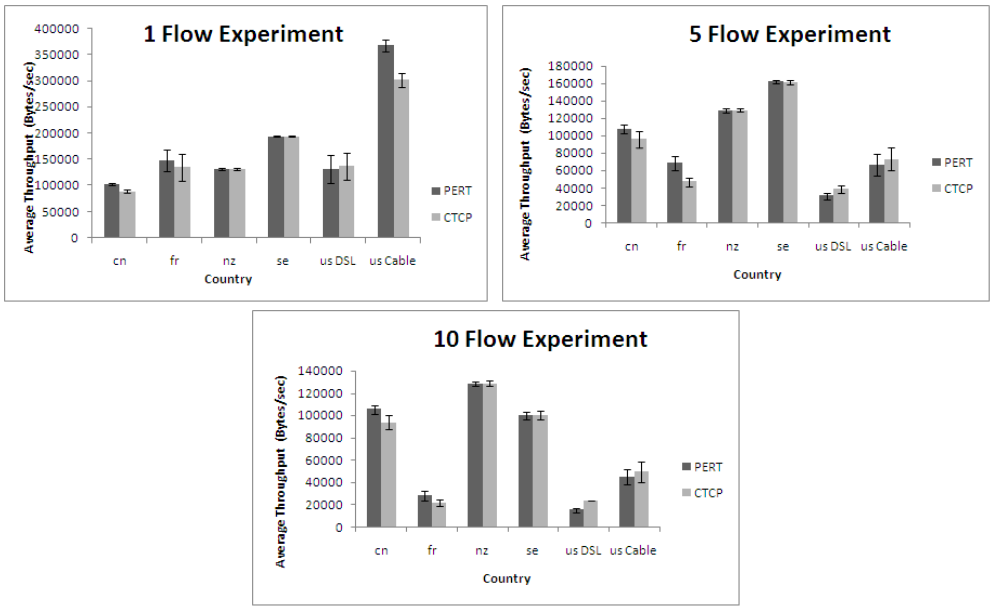

Fig. 16. Variation of throughput at different nodes across the world

US(Cable modem at College Station) respectively. We see from the figure that number of drops for PERT and CTCP is comparable with PERT having less number of losses in most of the cases, despite sharing the network links with numerous multiplexed flows, possibly using several different flavors of TCP. Both these graphs show that PERT can co-exist with the recently deployed and possibly future dominant protocol CTCP.

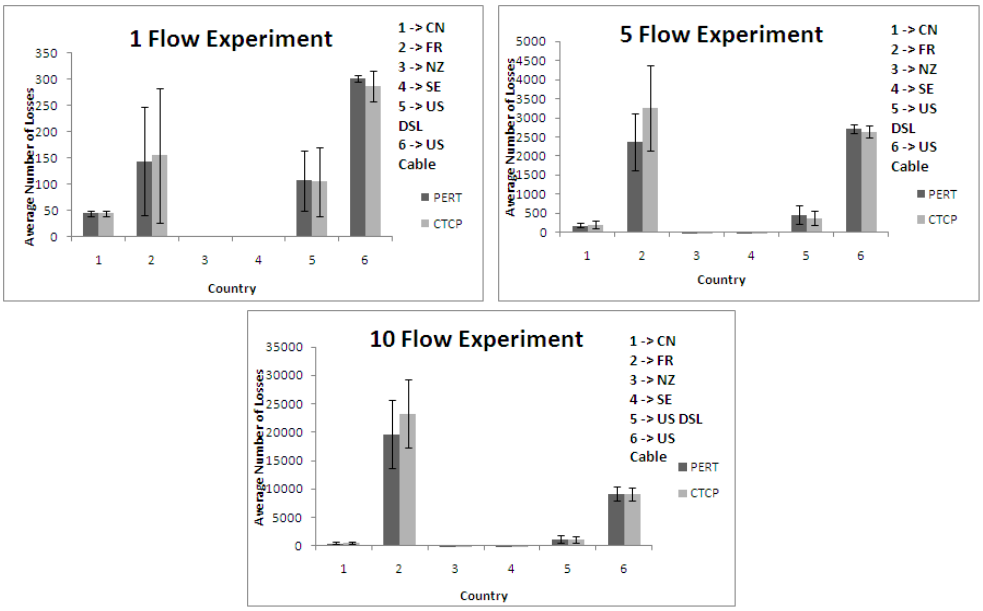

Fig. 17. Number of losses at different nodes across the world

CHAPTER VI

EVALUATION IN HOMOGENEOUS ENVIRONMENTS

Though our motivation for this work is to make PERT perform well in heterogeneous environments, the essence of the protocol is lost if, it does not perform well in homogeneous environments comparable to its original behavior. To study the behavior of modified PERT compared to the original PERT, we perform simulations similar to the ones in [6]. Almost all the simulations in [6] were performed. We present only the major results here to show that the modified PERT retains the desirable properties of the original PERT.

A. Impact of 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. 18 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. 18, 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.

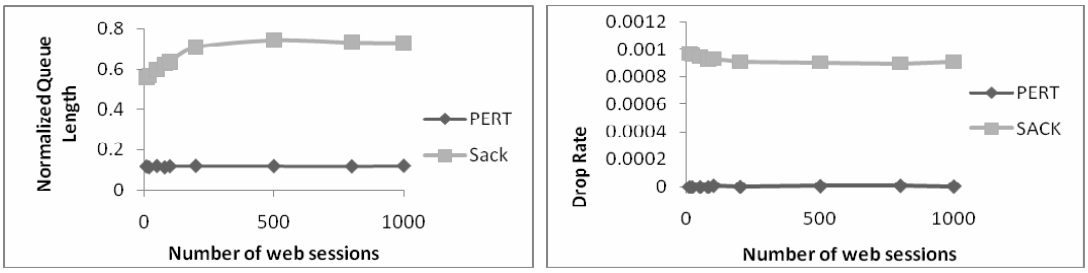

Fig. 18. Variation of normalized queue length and drop rate with the number of web sessions

B. Impact of number of long term flows

In another experiment, the number of long-term PERT flows is varied from 1 to 1000, keeping other parameters constant. Fig. 19 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 [6], in a similar experiment the performance of the original PERT was compared to SACK and it was observed that SACK observes high drop rates while PERT maintains zero loss rates. Also, we can see from Fig. 8 that in a 50-50 mix of PERT and SACK flows, the drop rate continuously raises, while PERT maintains zero loss rates while operating in homogeneous environments.

C. Impact of round trip delays

In another experiment, the end-to-end RTT is varied in the range of 10ms to 1 second. Fig. 20 shows the drop rate. From the figure, we see that the drop rate is non-zero when the delay is low, this is because with flows at lower RTTs PERT initially operates in high speed mode till the observed delays stay less than 5ms and gets easily pushed into the compete mode and operates aggressively again assuming

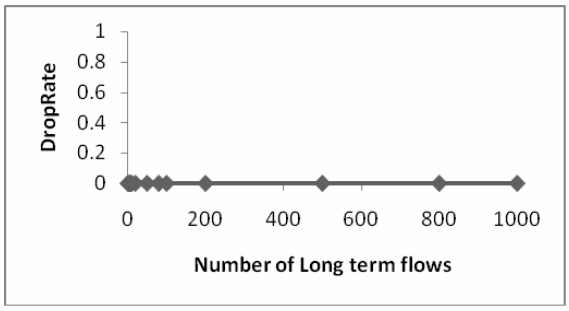

Fig. 19. Variation of drop rate with the number of long-term flows

that it is competing against loss-based protocols. We can see that the drop rate stays consistently zero for rest of the cases.

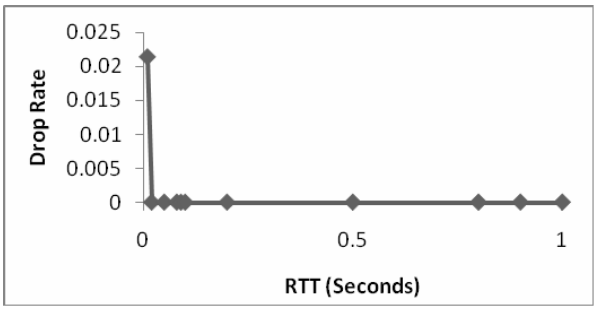

Fig. 20. Variation of drop rate with RTT

D. Multiple bottleneck simulations

In [6], PERT was demonstrated to perform very well in a multiple bottleneck link scenario described in the following way. The multiple bottleneck topology as shown in Fig. 21 consists of six routers labeled R1 to R6. The links between routers have a capacity of 150Mbps and a delay of 5ms. Each router is connected to a cloud of 20

nodes with a link of capacity 1Gbps and delay 5ms. The nodes in each cloud send data to the nodes in the cloud connected to the adjacent router. Also, all the nodes in the cloud connected to router R1 also send data to the nodes in the cloud connected to R6. Table I shows the average queue length, drop rate, utilization and Jain's Fairness Index of the link between each pair of routers as well as the Jain's Fairness Index of all the flows between each pair of routers. This is compared with the results in [6]. It can be noted that results do not vary much and the modified PERT still maintains low queue length and zero drop rates across all the bottleneck link queues. Moreover, the modified PERT behaves well in terms of drop rate (0) with a Jain's Fairness Index of 0.92 for the flows between R1-R6 over multiple bottlenecks. The link utilization and fairness in all other cases are comparable to the original PERT. In fact, the utilization gets better.

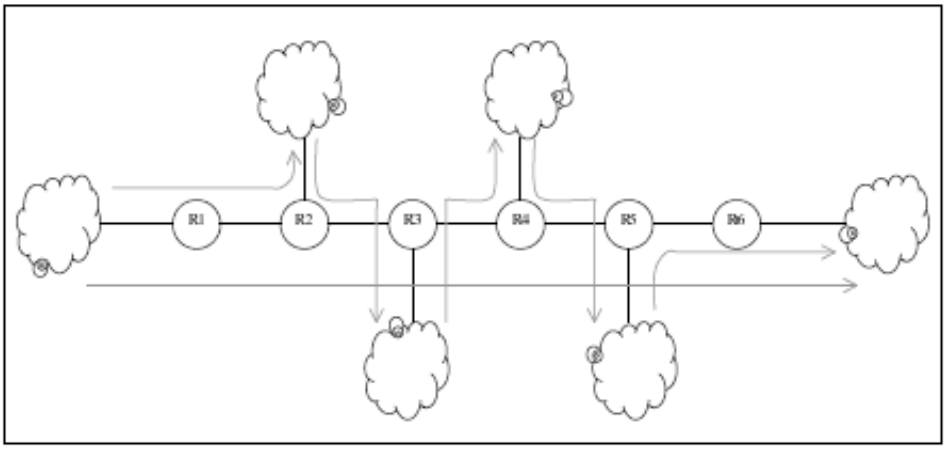

Fig. 21. Topology used for comparing the performances of original and modified PERT in a multiple bottleneck scenario

Table I. Drop rate, queue length, utilization and Jain's Fairness Index (JFI) of all flows between different links

Link	Drop rate	Normalized queue length	Utilization(Mbps)	JFI
R1-R2	0	0.172138	95.2394	0.999875
R2-R3	0	0.168135	95.238	0.999872
R3-R4	0	0.17454	95.2374	0.969581
R4-R5	0	0.172938	95.2385	0.999889
R5-R6	0	0.164932	95.2385	0.99986

E. Performance at low-buffers

Original PERT operates well at low-buffers [6]. To see how the modified PERT compares with the original PERT in terms of its performance at low-buffers, we conducted the following experiment. We had 20 flows sharing a bottleneck of 100 Mbps with an RTT of 60ms each. Buffer at the router is varied as a factor or delay-bandwidth product from a value of 1/128 to 1/2. Fig. 22 compares the utilization of PERT with that of original PERT. We can observe that the utilization of PERT is a little less at very low buffers. At larger buffer, in fact, the utilization improves with the modified PERT. It can be concluded that the performance of PERT at low buffers is comparable to that of original PERT.

All these results track the behavior of original PERT in homogenous deployment environment as reported earlier in [6].

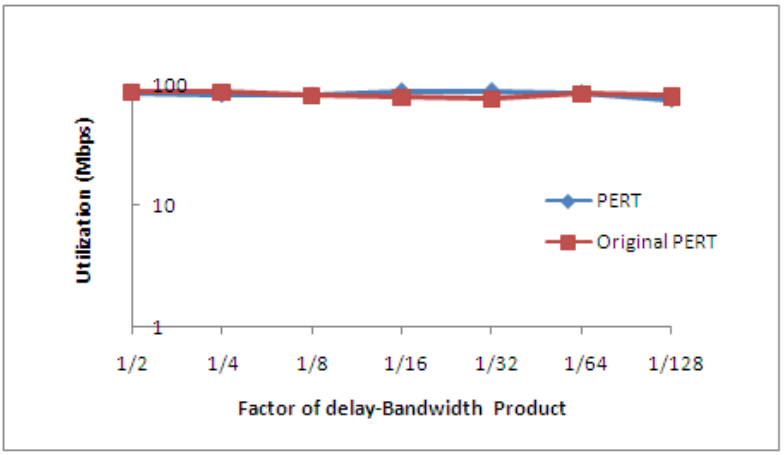

Fig. 22. Bottleneck link utilization at low-buffers

F. 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. 23 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 [11–13] which show significantly higher packet losses (several orders of magnitude difference) as shown in [11].

G. 4 flow convergence test

In all the earlier experiments with multiple flows, when all the flows were started at about the same time, very good intra-protocol fairness was exhibited. We now

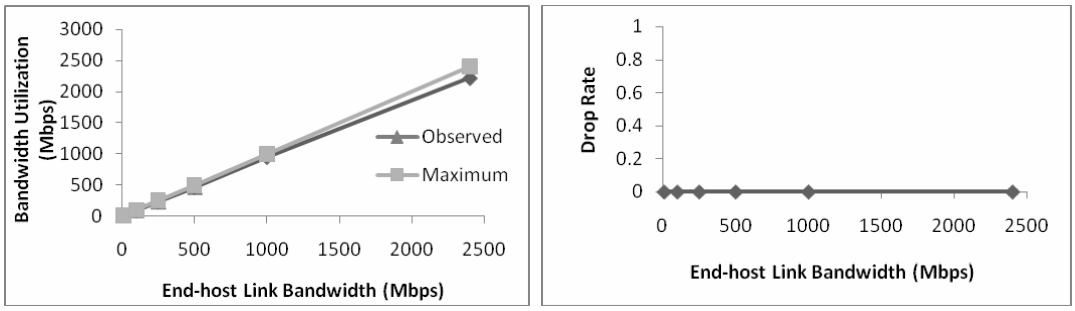

Fig. 23. Variation of utilization and drop rate at different end-host link bandwidths

evaluate the convergence properties of the modified PERT when flows start and stop at different times, dynamically changing the available link bandwidth. The first flow is started at time 0, and allowed to reach steady state. A new flow of the same flavor is then added every 300 seconds. The flows last for 2100, 1500, 900 and 300 seconds respectively. Fig. 24 shows the throughput of each flow over the time. From the graph we see that, PERT is capable of quickly increasing its sending rates and hence ensuring the link is fully utilized. It is also observed that the flows converge to fair share fairly quickly.

H. Robustness to noise

In this experiment, two flows with an end to end RTT of 60msshare a bottleneck bandwidth link with bandwidth of 1 Mbps. Uniform noise is generated with mean varying from 0 to 0.1 seconds 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 subject to noise. 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. 25 shows the link utilization

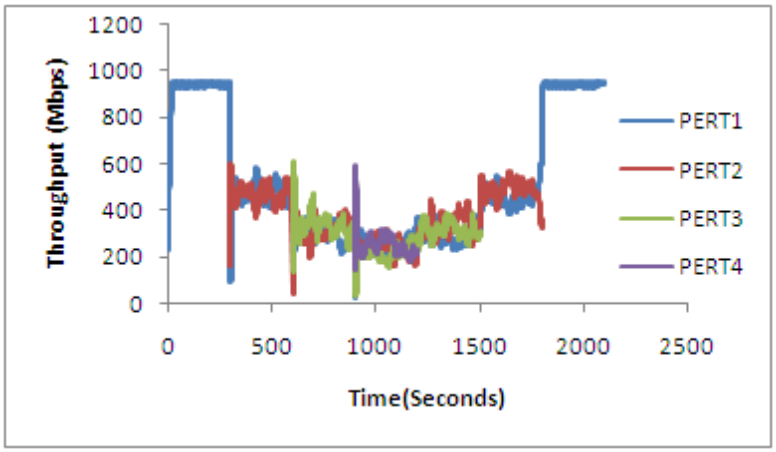

Fig. 24. Variation of drop rate with RTT

for different protocols as the mean is varied, for both the experiments. This shows that for PERT noise doesn't impact the correct deduction of congestion significantly and it maintains good Link utilization even at higher levels of noise.

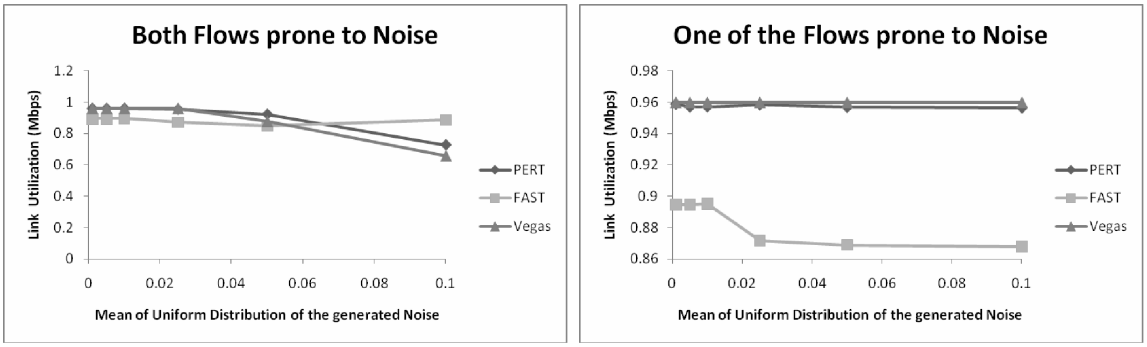

Fig. 25. Variation of link utilization with noise

I. 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 [8].

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. 26 plots the throughput against the random loss 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. 26. 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.

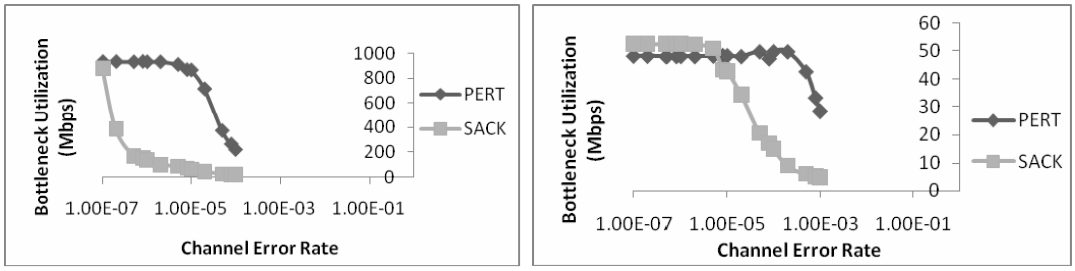

Fig. 26. Variation of bottleneck utilization with channel error rate in high-speed and normal wireless networks

CHAPTER VII

MISCELLANEOUS EXPERIMENTS

We also performed several other experiments to study the performance of PERT in miscellaneous scenarios. Each of them is described in detail and results are shown as follows.

A. Queue occupancy distributions with different mixes of protocols

Queue occupancy distribution is defined as the frequency of number of packets enqueued at a particular position, throughout the duration of the simulation in the router queue starting from position '1' to position 'queue length'. Queue occupancy distribution will give us an idea of how a particular protocol behaves in terms of distributing most of the packets at different positions in the queue, whether at the beginning of the queue (implying lower queuing delays meaning better performance) or at the end of the queue (implying higher queuing delays) or uniformly distributes packets throughout the queue.

We conducted experiments with our standard set up of 80 flows sharing a 150 Mbps link with an RTT of 60ms with the router buffer size set to the delay-bandwidth product of the link. With our setup, the maximum queue length comes to 1123 packets. We performed such experiments with flows of different protocols both individually and in a mix. Fig. 27 summarizes the results for the experiments with the same protocol. We can see from the figure that while with SACK most of the packets are enqueued at the end of the queue, with PERT most of them are enqueued at the beginning of the queue. We can also see that TCP-Illinois performs worse than SACK and that FAST distributes packets uniformly across all different positions in

the queue. Illinois' congestion behavior seems to keep queues longer than that of SACK as it tries to reduce window increments at higher delays. This clearly shows that PERT performs better in terms of queuing delay.

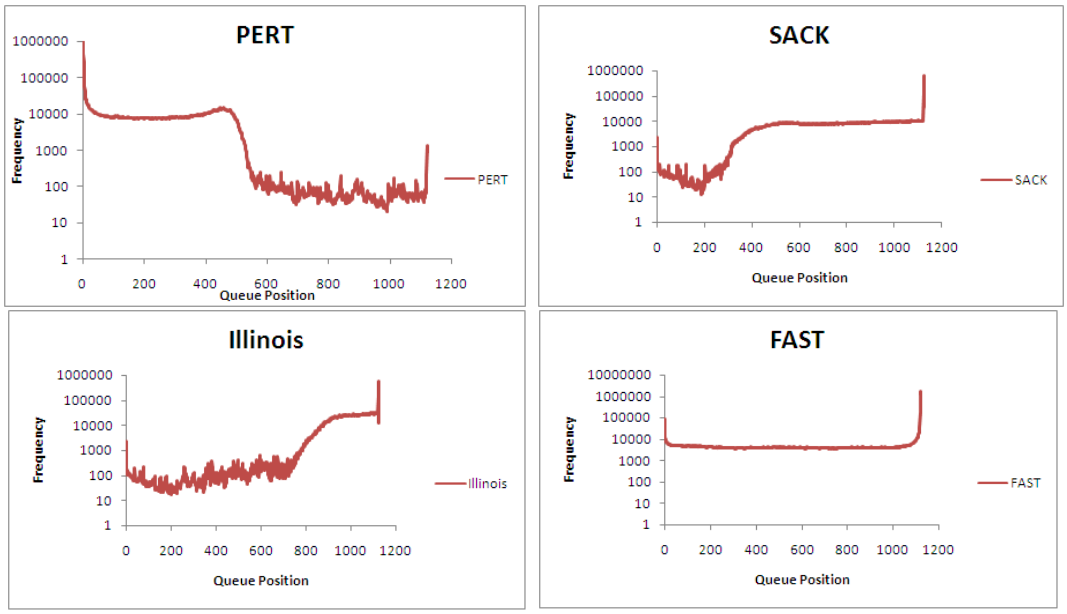

Fig. 27. Queue occupancy distributions with different protocols

We performed similar experiments with 50-50 mixes of different protocols like PERT-SACK, SACK-Illinois and FAST-SACK. Fig. 28 shows the results. We can see that, in the mix, PERT behaves similar to that of SACK in terms of number of packets queued at a particular location. This shows that PERT adapts to the scenario when it competes with SACK. We can also see that at the beginning of the queue, more PERT packets are enqueued compared to SACK and the end of the buffer more SACK packets are enqueued. Regarding mixes of SACK with other protocols, we can see that FAST clearly dominates SACK in terms of number of packets queued at different locations and hence kills the bandwidth share of SACK. In a mix of SACK

and Illinois, the queue occupancy seems to be somewhere between SACK and Illinois protocols alone.

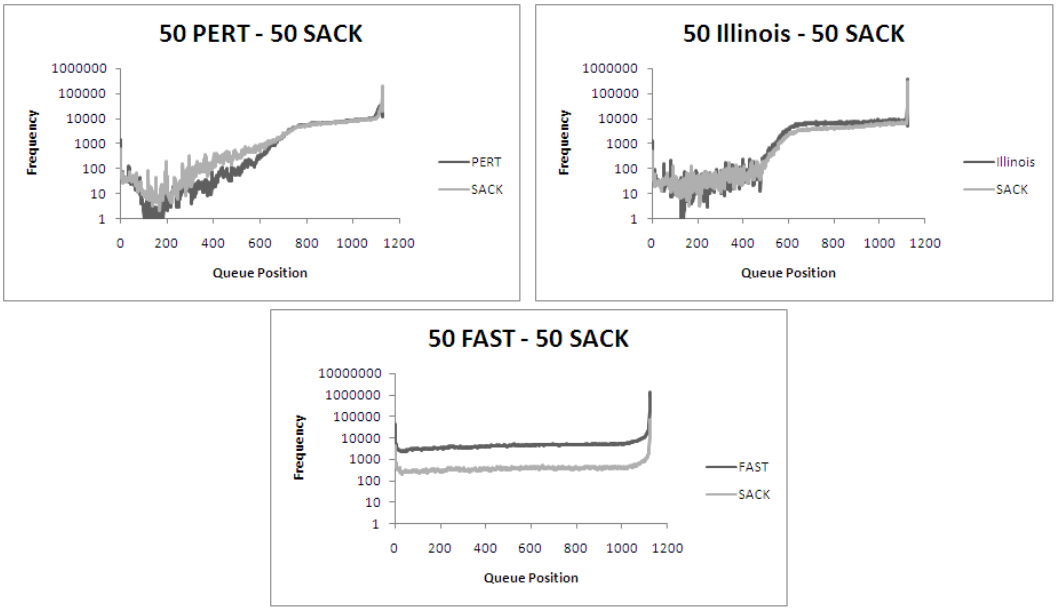

Fig. 28. Queue occupancy distributions with mixes of different protocols

B. Mixes of different protocols

We performed experiments with mixes of different protocols to see how they interact in a mixed environment. The set up for this experiment was similar to other mix experiments. 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. There were 99 flows in total competing for the bottleneck, with 33 flows of each of PERT, TCP-Illinois and TCP. Table II shows the results.

We can see that PERT performs the best among the three protocols in the mix

Table II. Bandwidth shares and drop rates of TCP-SACK, TCP-Illinois and PERT

Protocol	Bandwidth Share	Drop rate
SACK	34.5888	0.00275
Illinois	21.3449	0.00311
PERT	44.0663	0.00228

Table III. Bandwidth shares and drop rates of FAST TCP, TCP-SACK, TCP-Illinois and PERT

Protocol	Bandwidth Share	Drop rate
FAST	82.5992	0.00326
SACK	6.8721	0.0242
Illinois	6.5698	0.02499
PERT	3.9588	0.03598

both in terms of bandwidth share and drop rates. At the same time it is not very unfair in terms of bandwidth share. We performed a similar experiment including FAST-TCP [7], with 25 flows of each type in the mix. Table III shows the results. We observe that FAST behaves selfishly both in terms of bandwidth and drop rate gets more than 80% of the bandwidth share and forces other protocols to drop more packets.

C. PERT with RED

We conducted experiments to see how the modified PERT behaves when RED is used as queue management scheme at the router. In the experiment 20 flows share a bottleneck of 100 Mbps and the RTT of the all the flows was set to 60ms. The bottleneck buffer size was set to delay-bandwidth product. We performed similar ex-

periments with PERT-Droptail, SACK-RED, SACK-Droptail. Table IV summarizes the results. We can see from the table that though RED has an effect in lowering the average queue length, the drop rate gets worse for PERT when RED is used. This is because, PERT gets a wrong impression about the congestion at the router and always operates in the high-speed region, where it sends many packets. because of higher α employed in this region, PERT experiences higher packet loss rate. The observed loss rate is higher than that of SACK with RED. Whereas, when Droptail is applied at the router, it judges the congestion correctly and operates in the correct region (*i.e.,* the stable non-aggressive region) leading to zero loss rate.

Table IV. Drop rate and normalized average queue length for PERT and SACK with different queue management schemes at the router

Scheme	Normalized queue length	Drop rate
PERT-RED	0.0214	0.028
PERT-Droptail	0.083	0
SACK-RED	0.016	0.012
SACK-Droptail	0.576	0.0004

D. PERT with intermittent TCP

We performed another experiment where we had 80 PERT flows start at time 0 and share a bottleneck link of 150Mbps. We Had equal number of TCP flows start at 100 seconds and go away at 200 seconds. PERT flows stay on until 300 seconds. The interesting observation that we made with this experiment is that though initially PERT flows operate in the safe region (as they compete among themselves) and later get pushed into the compete region (as the TCP flows start competing for the bandwidth at 100 seconds), the flows do not get pushed back into the safe region once

the TCP flows goes away. This is because, once the PERT flows start operating in the aggressive (compete) mode, they contribute to the increase in the queue size and even after TCP flows go away, they still operate in the compete region assuming that they are competing with TCP, even when they operate alone. Table V shows the normalized average queue length and drop rates at different points of time. We can see that initially at 100 seconds when there are no SACK flows, the normalized queue length is low and drop rate is 0. As SACK flows also compete for the bottleneck at 200 seconds, the drop rates and the average queue length increase. However, at 300 seconds even when SACK flows go away and PERT contends alone for the bottleneck, the average queue length and drop rate still remain high, because PERT still operates in the compete region as mentioned earlier.

Table V. Drop rate and normalized average queue length at the router at different points of time in the experiment

Time(Seconds)	Normalized queue length	Drop rate
100	0.113	0
200	0.760	0.0196
300	0.753	0.0190

In the same experiment, to see when the PERT flows get back to the safe region after TCP flows go away, we let each PERT flow exit at every 10 seconds. We observed that PERT flows get back to their original operating point (*i.e.,* safe region) when about 65 PERT flows exit *i.e.,* when 20% of initial number of PERT flows operate. Similarly, to see with how many number of TCP flows it takes for PERT to enter into high delay region, we added 1 TCP flow, every 10 seconds after initial 100 seconds (when PERT operates alone). We observed that PERT flows enter the compete region

when the number of TCP flows added equals about 9 flows that is 10% of TCP flows. This means that PERT flows could tolerate up to 10% of TCP flows to get back to safe region once the TCP flows leave.

E. CPU utilization with different congestion protocols

We performed experiments with the kernel implementation of TCP, CTCP and PERT to measure the CPU utilization when flows of different protocols operate. One of our machines in the lab acted as a server and a China based planet-lab machine acted as a client. We performed experiments with different number of concurrent flows from the client. With 1,5,10 and 25 flows from the clients, there wasn't any significant value for CPU utilization. However, with 50 flows the values were notable. Table VI shows the CPU Utilization for TCP-SACK, CTCP and PERT.

Table VI. CPU utilization with different congestion protocols with 50 flows from the client

Protocol	CPU utilization Range(%)
SACK	1.5 - 3.5
CTCP	1 - 3
PERT	2 - 3.6

It can be noted from the table that with these three different protocols similar CPU utilization is observed, while with PERT the CPU utilization is a little higher.

CHAPTER VIII

CONCLUSIONS AND FUTURE WORK

In this research, we have identified the issues in adapting a delay-based protocol to heterogeneous environments, especially in making it co-exist with flows of flavors of dominant protocols in the Internet. We presented a rationale and design modifications for adapting the delay-based protocol PERT to work well in heterogeneous environments. With the suggested design modifications, we also preserve its original properties when operating alone.

While, there are some existing protocols like TCP-Illinois and Microsoft's Compound TCP (CTCP), they are loss-based and do not perform as well as PERT while operating alone. The delay-based protocol FAST can compete with loss-based protocols. But it does not provide incentives for incremental deployment, as we show. Further, 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 especially the delay-based ones and providing incentives (lower packet drop rates, in our case) for new protocol deployment. We support our claims through extensive simulations of a wide number of practical scenarios in heterogeneous environments using *ns-2*. We have also tested and compared PERT with SACK in terms of bandwidth share and packet losses, 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 protocols performance and that PERT gets higher bandwidth and lower drop rates while giving the competing flows

a fair bandwidth share.

We observe that PERT not only co-exists with other protocols in wide range of scenarios, but also serves as a high speed protocol in terms of its performance in high bandwidth-delay networks and has comparable performance similar to high speed protocols such as LTCP, while operating alone. We also observed that PERT can tolerate channel errors more gracefully than SACK.

We have identified the problems with PERT and are aware of scenarios in which PERT does not perform very well. We intend to fix the known issues as a part of our future work. We also plan to investigate the issues in improving global network characteristics as a result of incremental deployability of new delay-based protocols such as PERT in the future.

REFERENCES

[1] S. Floyd and V. Jacobson, "Random early detection gateways for congestion control," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-412, August 1993.

[2] S. Athuraliya, V. H. Li, S. H. Low, and Q. Yin, "REM: active queue management," *IEEE Network*, vol. 15, no. 3, pp. 48-53, May/June 2001.

[3] L. Brakmo, S. O'Malley, and L. Peterson, "TCP Vegas: new techniques for congestion detection and avoidance," in *Proc. SIGCOMM '94 Symposium.*, 1994, vol. 24, no. 4, pp. 24-35.

[4] R. S. Prasad, M. Jain, and C. Dovrolis, "On the effectiveness of delay-based congestion avoidance," in *Proc. Second International Workshop on Protocols for Fast Long-Distance Networks*, Argonne, IL, February 2004, pp. 3-4.

[5] S. Rewaskar, J. Kaur and D. Smith, "Why dont delay-based congestion estimators work in the real-world?," Technical Report TR06-001, Department of Computer Science, University of North Carolina Chapel Hill, July 2005.

[6] S. Bhandarkar, A. L. N. Reddy, Y. Zhang, and D. Loguinov, "Emulating AQM from end hosts," *SIGCOMM Comput. Commun. Rev.*, vol. 37, no. 4, pp. 349-360, October 2007.

[7] C. Jin, D. X. Wei, and S. H. Low, "FAST TCP: motivation, architecture, algorithms, performance," in *Proc. IEEE INFOCOM 2004*, vol. 4, Mar. 2004, pp. 2490-2501.

[8] S. Liu, T. Basar and R. Srikant, "TCP-Illinois: a loss and delay-based congestion control algorithm for high-speed networks," in *Proc. 1st International Conference on Performance Evaluation Methodologies and Tools*, Pisa, Italy, October 11 - 13, 2006, pp. 55.

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

[10] M. Dischinger, A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing residential broadband networks," in *Proc. 7th ACM SIGCOMM Conference on Internet Measurement*, San Diego, CA, October 24 - 26, 2007, pp. 43-56.

[11] S. Bhandarkar, S. Jain, and A. N. Reddy, "LTCP: improving the performance of TCP in highspeed networks," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 1, pp. 41-50, Jan. 2006.

[12] L. Xu, K. Harfoush, and I. Rhee, "Binary increase congestion control for fast long-distance networks," in *Proc. IEEE INFOCOM 2004*, vol. 4, Mar. 2004, pp. 2514-2524.

[13] D.J. Leith, and R. Shorten, "H-TCP protocol for high-speed long distance networks," in *Proc. Second International Workshop on Protocols for Fast Long-Distance Networks*, Argonne, IL, February 2004.

[14] S. Floyd, "HighSpeed TCP for large congestion windows," RFC 3649, December 2003.

[15] W. Feng, D. Kandlur, D. Saha, and K. Shin, "A self-configuring RED gateway," in *Proc. INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Com-*

puter and Communications Societies, New York, NY, March 1999, vol. 3, pp. 1320-1328.

[16] D. Bansal, and H. Balakrishnan, "Binomial congestion control algorithms," in *Proc. INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies*, Anchorage, AK, 2001, vol. 2, pp. 631-640.

[17] D. Bansal, H. Balakrishnan, S. Floyd, and Scott Shenker, "Dynamic behavior of slowly-responsive congestion control algorithms," in *Proc. 2001 ACM SIGCOMM*, San Diego, CA, August 2001, pp. 263-274.

[18] G. Appenzeller, I. Keslassy, and N. Mckeown, "Sizing router buffers," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 4, pp. 281-292, August/September 2004.

[19] Y. Zhang, "Modeling and stability of PERT," Technical Report TAMU-ECE-2007-03, Texas A&M University, May 2007.

[20] S. Floyd, M. Handley, J. Padhye, and J. Widmer, "Equation-based congestion control for unicast applications," *SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 43-56, August 28 - September 01, 2000.

[21] P. Sarolahti and A. Kuznetsov, "Congestion control in linux TCP," in *Proc. FREENIX Track: 2002 USENIX Annual Technical Conference*, Philadelphia, PA, June 2002, pp. 4962.

VITA

Kiran Kotla received his B.E.(Hons) degree in computer science from the Birla Institute of Technology and Science (BITS), Pilani, India in 2005 and his M.S. in computer enginnering from Texas A&M University, College Station, TX, in 2008. His research interests are in the areas of networking congestion protocols and wireless ad hoc networks. He has been a recipient of the BITS Merit cum Need Scholarship from the BITS, Pilani, India during 2001-2005 and Industry Affiliates Program (IAP) scholarship sponsored by Hewlett Packard at the Department of Computer Science, Texas A&M University. Prior to arriving at Texas A&M, he worked in the Infrastructure Products Group of Juniper Networks, India from 2005-2006. He can be contacted at the following address : Department of Computer Science, Texas A&M University, 301 HRBB, College Station, TX 77843-3128.

The typist for this thesis was Kiran Kotla.