On Randomizing the Sending Times in TCP and other Window Based Algorithms

K. Chandrayana, S. Ramakrishnan, B. Sikdar, S. Kalyanaraman, A. Balan, O. Tickoo
Department of ECSE, Rensselaer Polytechnic Institute
Troy, NY 12180

Abstract— **Current implementations of TCP suffer from performance problems like bias against flows with higher round trip times (RTTs), synchronization of windows, phase effects in flows and correlated losses leading to throughput degradations with Drop Tail queues. In this paper we propose a solution to these issues by introducing randomization into the network in the form of *Randomized TCP*. Instead of sending back-to-back packets, Randomized TCP spaces successive packet transmissions with a time interval $\Delta = RTT(1+x)/cwnd$, where x is a zero mean random number drawn from an Uniform distribution. We show that the proposed scheme is better than Paced TCP as well as TCP Reno in terms of throughput, fairness, loss rates, timeouts and latency in various single and multi-bottleneck scenarios. We also show that Randomized TCP reduces phase effects and synchronization even when multiplexed with TCP Reno. Also, we extend randomization to other window based schemes like the Binomial schemes and show that fairness to TCP Reno increases dramatically. Network wide benefits like fairness, reduction of phase effects, synchronization and burst drops can be obtained even when only a subset of the flows are randomized thereby motivating incremental deployment.**

Keywords— **TCP, flow synchronization, fairness, phase effects**

I. Introduction

Current TCP implementations have been known to suffer from a number of phenomena which limit their effectiveness and degrade performance, the primary amongst them being: synchronization of congestion windows (and thus the loss instances) causing alternate overloading and underloading of the bottleneck [12], [14], [15]; phase effects wherein a certain section of flows face recurrent losses [6]; unfairness to flows with higher RTTs [5]; delays and losses due to the bursty nature of TCP traffic [15], [2]. In this paper we look at a comprehensive solution to all these issues by randomizing the packet transmission times in TCP flows.

Synchronization of windows and loss events for flows sharing common links causes alternating periods of overload and underload thereby leading to inefficient resource utilization. Synchronization can be attributed to two reasons: (1) the sliding window flow control of the TCP, which produces bursts of packets and (2) the Drop Tail queue at the bottleneck, which drops all packets when the buffer is full [8]. *Phase effects* refer to conditions where in the bandwidth-delay product of the path of a flow is not an integral multiple of the packet size [6]. Phase effects cause a specific section of competing flows to experience recurrent drops causing unfair distribution of bandwidth and increased latency. Phase effects are manifested in the network preferentially dropping packets from a specific subset of flows thereby reducing their throughput.

Random Early Drop or RED [7], tries to solve the problem of full queues, flow synchronization and phase effects by dropping packets probabilistically if the queue length is above some threshold thereby avoiding burst losses. However, it was shown in [11] that the number of consecutive drops is higher with RED than Tail Drop, suggesting RED might not help as anticipated with the synchronization of flows. TCP Pacing was proposed in [15] to solve the problem of bursty losses by pacing the packet transmission times at the sender. In [2], it is shown that while pacing reduces synchronization for long flows, short Paced TCP flows get synchronized. Also, it was shown that paced TCP cannot compete fairly when placed together with TCP Reno flows. The effect of pacing in solving the phase effects has not been investigated in literature.

The basic idea behind our scheme is that introducing randomization into the system can break synchronization. Also by introducing the randomization, we avoid burst losses, thereby making the loss events "distributed". This then helps in solving the problem of phase effects. Though RED can introduce randomization in networks to some extent, it is not widely deployed due to variety of reasons [10], [**?**]. In this paper, we propose a modification to TCP, called *Randomized TCP*, as a mechanism for introducing randomization into the network. In Randomized TCP, instead of sending back to back packets, the packet sending times are randomized. Specifically, successive packets of a window are sent after an interval of $RTT(1+x)/cwnd$,

Supported in part by ARO contract DAAD 19-00-1-0559 and NSF contract ANI 9806660

where $cwnd$ is the congestion window in packets and x is a random number drawn from an Uniform distribution on $[-I, I]$. We call I the randomization interval. This solution is distributed, can be implemented at the end systems and has just a single parameter to be configured, which makes it very attractive from an implementation point of view.

The increased randomization increases the entropy of the system which correspondingly reduces the queue sizes thereby improving the stability of the system [18]. Our results show that Randomized TCP performs better than or as well as Paced TCP and TCP Reno, independent of the capacity and buffer size at the bottleneck and for both short and long flows. The performance improvements can be seen in throughput, fairness, loss rates, timeouts and latency of the flows. We also show that Randomized TCP reduces phase effects and synchronization even when multiplexed with TCP Reno flows. We also investigate the impact of randomization on a class of slowly varying congestion control schemes called Binomial schemes [3]. We show that by incorporating randomization in these schemes, the fairness increases dramatically when competing with TCP flows in drop tail queues. In other words, our scheme can emulate the beneficial effects of RED in a distributed manner without the complexities and unfavorable aspects of RED. In addition, the benefits of randomization can be reaped even when it is partially deployed.

In short, the main contributions and conclusions of this paper are as follows:

- It proposes a distributed mechanism to introduce randomization in network flows to prevent synchronization and phase effects
- Phase effects with Drop Tail queues which persist with TCP Reno are removed if Randomized TCP is used
- The presence of a single Randomized TCP flow can break the synchronization and phase effect at the bottleneck motivating incremental deployment
- Randomized TCP tries to distribute losses over time, emulating the beneficial aspects of RED
- Randomized TCP reduces the bias against flows with longer RTTs in Drop Tail queues
- Randomization considerably improves fairness of binomial schemes even with Drop Tail queues

The rest of the paper examines the Randomized TCP in detail. In Section II we discuss previous work in this area. Section III describes the Randomized TCP scheme, the intuitive reasons as to why it works and the evaluation of an optimal randomization interval. Section IV describes the implementation details, performance metrics and the simulation setup. Comparison of the performance of Randomized TCP, Paced TCP and TCP Reno is described in Section V while Section VII evaluates the phase effects, synchronization and burst losses for Randomized TCP and TCP Reno. Section VIII compares the performance of Randomized Binomial schemes and Section IX describes the Linux implementation of Randomized TCP. Finally we present the conclusions and future work in Section X.

II. Background and Related Work

Rate based schemes, in contrast to window based schemes, send out packets at regular intervals thus avoiding burst transmissions. Since rate based schemes loosely observe the packet conservation principle they can at times be less responsive to network congestion. TCP Pacing [15] is a hybrid approach between window based schemes and rate based schemes. In pacing, packets to be sent in a window are spaced by $\Delta = RTT/cwnd$. This spacing of packets avoids back to back transmissions and hence removes the burstiness of TCP.

Pacing was first suggested in [15] as a correction for the compression of ACKs due to cross traffic. Since then the concept of pacing has been applied to slow-start, after a packet loss and after an idling time in case of web traffic. For details on TCP pacing, we refer the reader to [2] and the references therein. In [2] it is shown that with long flows Paced TCP removes synchronization, improves fairness over TCP Reno and achieves the same throughput as TCP Reno. However, in presence of short flows the authors show that Pacing gets synchronized causing larger latencies. Also, the authors show that when Paced TCP competes against TCP Reno, it gets an unfair share of the bandwidth.

A modified version of pacing is also evaluated in [9]. In [9] the spacing interval is defined as $\frac{RTT}{cwnd+V}$, where V is the tunable parameter, which controls the aggressiveness of the pacing. However, the effect of this scheme on the synchronization of flows is not investigated. They observe that with bulk data transfer the modified pacing shows results similar to TCP Reno. However, for a web-like load model, the modified paced TCP exhibits lower packet loss than TCP and also the average transfer latencies are lower. The authors, however do not discuss the parameter setting for V and it's effect on the pacing scheme. Also, they do not consider the case where TCP Reno and Paced TCP are multiplexed on the same link.

III. Randomized TCP

Randomized TCP is similar to Paced TCP in that it "paces" packet transmissions but instead of spacing the transmissions equally, it adds or subtracts a random interval to the packet sending times at TCP sources. Packet transmissions are scheduled at intervals of $\frac{RTT}{cwnd}(1 + x)$,

where x follows the Uniform Distribution on $[-I, I]$. Evidently, I has to be between 0 and 1. A packet is transmitted at the expiry of the timer, if the window allows a packet to be sent. If not, upon reception of an ACK, we schedule the packet transmission with a random delay of $\frac{RTT}{cwnd}y$, where y is $U(0, I)$. Setting I to 0 reduces Randomized TCP to Paced TCP.

In Section V, we investigate the optimal setting of the randomization interval and find that a Uniform distribution on $[-1, 1]$ is the best. This choice of Uniform distribution can be intuitively justified as; *a)* since the distribution is centered around *0*, on an average there is "no randomization" and Randomized TCP behaves as Paced TCP, *b)* and a minimum value of *-1* of randomization implies TCP Reno implementation. This implies that sometimes we send back-to-back packets and sometimes we send paced packets. Thus with a randomization interval value of 1, randomized TCP keeps moving forth between TCP Reno and TCP Paced. Intuitively, this entails an early detection of congestion (when the TCP behaves as Reno) and an even distribution of losses and throughput (when TCP behaves as Paced). Thus Randomized TCP takes the best of both Reno and Paced TCP and ensures lesser drops (because of early congestion detection) and fairer throughput.

In Paced TCP packets from each source reach the bottleneck at an uniform rate which can lead to near perfect interleaving. Such situations can cause all sources to lose packets thereby resulting in all the sources decreasing their windows together, resulting in synchronization. But with randomization, the rate is not uniform at the bottleneck and packets from flows are dropped after differing times due to the extra delay incurred due to randomization. This means that sources decrease their windows at different times and hence the periods of increase and decrease are not as synchronized as in Paced TCP. Also due to non-uniform rate a single source may lose two packets while some other source may not lose packets at all for the time being. So the congestion epochs for different flows get out of sync and the network utilization is higher. But the nice property that comes because of randomization is that the source which has lost packets once is very less likely to lose again (this may not be the case with deterministic TCP for some parameter settings [6]), thereby ensuring that over a larger time scale the rate distribution is fair. We also note that the probability of two packets coming nearly back to back is significant only when the window size is large. This means that the probability of multiple packet drops will be very low if the window size is small, thereby reducing timeouts.

Randomizing the sending times results in extra delays causing the RTT to increase artificially. This causes Randomized TCP to get beaten down when competing with TCP Reno. It is well known that TCP's throughput is directly proportional to the square root of the window increase parameter and inversely proportional to RTT [13]. To allow Randomized TCP to compete fairly with TCP Reno, we analytically characterize the increased RTT and make the increase factor in TCP proportional to the square of the ratio of the changed RTT to the real RTT.

A. Analytical Characterization of Increase Parameter for Randomized TCP

In this section we outline the methodology for setting the increase parameter, α for Randomized TCP so as to make it compete fairly with TCP Reno. This is required because randomizing the sending times results in extra delay and hence slows down the window growth. As such it is likely that Randomized TCP will lose to TCP Reno when competing on a single bottleneck.

Consider a Randomized TCP connection with a constant window size of *w*. Let the real RTT for the connection be a constant denoted by *R*. Each packet is sent after a time equal to $R(1 + x)/w$ where x is a Uniform random variable between $[-I, I]$ (The optimal value of this interval is shown to be 1 in section V, but presently we treat it more generally). Let the first packet be sent at time $t = 0$. Then the timer for the $w + 1^{th}$ packet of the connection will be scheduled at time, say t_1, such that

$$t_1 = R(1 + \frac{1}{w}\sum_{i=1}^{w} x_i) \tag{1}$$

where x_i is the random value for the i^{th} packet in the window. The x_is are independent and identically distributed. The effective RTT of the flow is the given by the time when $(w + 1)^{th}$ packet is sent. In the absence of random variations in real RTT, the ACK for the first packet comes exactly after time R. If $\sum_{i=1}^{w} x_i \geq 0$ then $t_1 > R$ and we will send the $(w + 1)^{th}$ packet at time t_1. Else, the $(w + 1)^{th}$ packet will be sent after a random time $\frac{RTT}{w}y$ after the ACK arrival, where y is drawn from an uniform distribution on [0,I].

Thus the effective RTT can be expressed as

$$RTT_{eff} = \begin{cases} R(1 + \frac{1}{w}\sum_{i=1}^{w} x_i) & \text{w.p. } P\{\sum_{i=1}^{w} x_i \geq 0\} \\ R(1 + \frac{y}{w}) & \text{w.p. } P\{\sum_{i=1}^{w} x_i \leq 0\} \end{cases} \tag{2}$$

where w. p. is short for "with probability". Then, the mean effective RTT, $\overline{RTT}_{eff}$, can be expressed as

$$\begin{aligned} \overline{RTT}_{eff} &= \{R(1 + \frac{1}{w}E[\sum_{i=1}^{w} x_i \mid (\sum_{i=1}^{w} x_i \geq 0)])\} \\ & P\{\sum_{i=1}^{w} x_i \geq 0\} + \{R(1 + \frac{\bar{y}}{w})\}\ P\{\sum_{i=1}^{w} x_i \leq 0\} \end{aligned} \tag{3}$$

where $\bar{y}$ is the mean of y equal to $I/2$. Since x_i follows an Uniform distribution around zero, its easy to see that $P\{\sum_{i=1}^{w} x_i \geq 0\ \} = P\{\sum_{i=1}^{w} x_i \leq 0\ \} = 0.5$.

Assuming that the window size is sufficiently large to invoke the the Central Limit Theorem we get

$$\sum_{i=1}^{w} x_i \sim N(0, \sigma^2) \tag{4}$$

where

$$\sigma^2 = w * \frac{I^2}{3} \tag{5}$$

The pdf of $\sum_{i=1}^{w} x_i$ conditioned on $\sum_{i=1}^{w} x_i \geq 0$ can be found out to be twice that of the Gaussian pdf multiplied by the Unit step function. From this we can derive the conditional mean as

$$E[\sum_{i=1}^{w} x_i \mid (\sum_{i=1}^{w} x_i \geq 0)] = \sqrt{\frac{2wI^2}{3\pi}} \tag{6}$$

Plugging these back into the equation for $\overline{RTT}_{eff}$, we obtain

$$\overline{RTT}_{eff} = R + \frac{1}{2w}(\sqrt{\frac{2wI^2}{3\pi}} + \frac{I}{2}) \tag{7}$$

For Randomized TCP with increase parameter α and effective mean RTT, $\overline{RTT}_{eff}$, the throughput is proportional to $\frac{\sqrt{\alpha}}{\overline{RTT}_{eff}}$. To make the throughput same as that of TCP Reno (with $\alpha = 1$ and $RTT = R$), we set $\alpha = \frac{\overline{RTT}_{eff}^2}{R^2}$ for randomized TCP. In the real implementation, since window value changes with time, $\overline{RTT}_{eff}$ changes with time and so we change the value of α also with time.

IV. Implementation and Simulation Setup

We have implemented Randomized TCP in the network simulator *ns*. For our implementation, we used the congestion control and loss recovery mechanisms of TCP Reno and thus Randomized TCP has the usual slow-start and fast recover and retransmit mechanisms. For the simulations reported in this paper, we disabled the delayed acknowledgments option. Also we used the modified window increase parameter for Randomized TCP implementation.

Figure 1 shows the topology used in the simulations. The access links were configured at a rate 4 times greater than that of the bottleneck link. All the links use Drop Tail queues unless otherwise specified. Default settings for the simulations are a round-trip time of 100ms, a bottleneck bandwidth of 4Mbps and a buffer of 25 packets. The maximum advertised window is set sufficiently high so that it does not constrain the actual window. We use a Maximum Segment Size of 500 bytes.

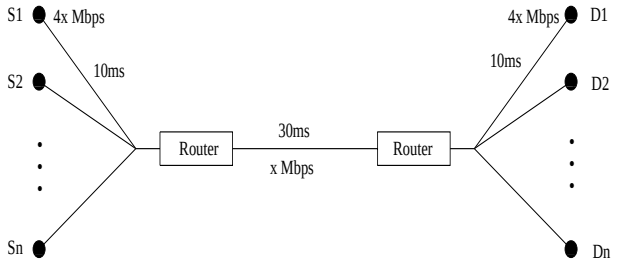


Fig. 1. Topology used in the simulation.

We evaluate the performance of randomized TCP for the following set of metrics: average throughput, fairness, loss rates, timeouts, latency and synchronization. We characterize fairness using the modified Jain's fairness index, [4], [2]. Jain's fairness index is defined as

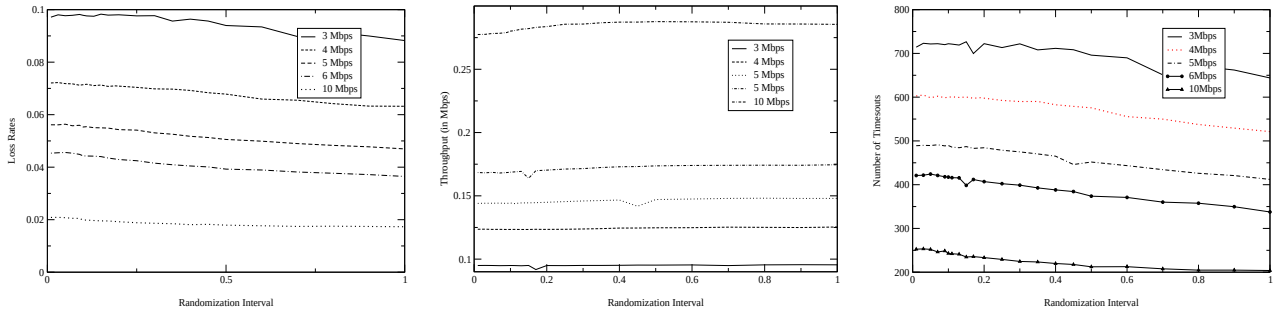
$$f = \frac{(\sum_{i=1}^{n} x_i.RTT_i)^2}{n(\sum_{i=1}^{n}(x_i.RTT_i)^2)} \tag{8}$$

where x_i is the throughput of the i^{th} flow, RTT_i is the round-trip time of flow i and n is the number of competing flows.

To study the synchronization of flows we use the covariance between the congestion window of two competing flows. Flows would be synchronized if their windows increase and decrease simultaneously. In this case both flows' windows (say w_1 and w_2) would be above or below their mean values at any time t, i.e. $(w_1(t) - \bar{w_1})(w_2(t) - \bar{w_2}) > 0$. So the cross-covariance coefficient of synchronized flows would be positive. In the case where the flows are totally out of sync, $(w_1(t) - \bar{w_1})(w_2(t) - \bar{w_2}) < 0$, since when one flow has a large window, the other would have a smaller window and vice versa. So the cross-covariance coefficient of out of sync flows would be negative. This shows that the cross covariance coefficient of greater than 0 implies in-phase synchronization while less than 0 implies out-of phase synchronization. However, too large a negative value of cross-covariance denotes that synchronization effects still persist albeit in a negative sense. In [15] the authors also argue that out-of-phase synchronization is not good. Hence a value equal to or close to 0 for cross-covariance coefficient should be the optimal.

In the following sections we present the simulation results. We first, observe the effect of bottleneck bandwidth, buffer sizes and RTTs on our tunable parameter, the randomization interval I in section V. Using these simulations we propose a value of the interval for optimal performance.

We then evaluate the performance of Randomized TCP with both bulk and short transfers. For bulk transfers we consider two cases, (a) when all the flows have the same RTT and (b) when each flow has a different RTT



(a) Loss rate (b) Throughput (c) Timeouts

Fig. 2. Loss Rate, Throughput and Timeouts for 30 flows as a function of randomization interval, for different values of bottleneck bandwidth.

and the results are reported in Section VI-A. Randomized TCP's performance for Web like transfers is investigated in sectionVI-B. For this case we vary the transfer load from 10 packets to 2500 packets. In Section VI-C, we investigate the interaction between Randomized TCP and Reno, for fairness, throughput, losses and timeouts.

Section VII shows the performance of Randomized TCP with respect to phase effects, synchronization amongst flows and burst losses. In Section VII-A we show the reduction in phase effects with Randomized TCP for both single and multiple bottleneck topology. We also calculate the degree of synchronization in Section VII-B, using covariance coefficients, of Randomized , Reno and Paced TCP for both Bulk and short data transfer. Section VII-C describes the reduction in burst losses with Randomized TCP over TCP Reno. In Section VIII we extend the randomization to Binomial Congestion Control Algorithms and evaluate the fairness properties of Randomized Binomial schemes competing with Randomized TCP.

V. Parameter Tuning

The randomization interval has a significant impact on the performance of Randomized TCP, and hence its characterization is of utmost importance. In this section we study the effect of change in bottleneck bandwidth, buffer size, number of flows and round-trip times on throughput, number of losses, timeouts as a function of the randomization interval. Through these simulations we obtain the optimal value of randomization interval. The default settings for this section are a bottleneck link of 1 Mbps, all the other links of bandwidth 4 Mbps, end-to-end propagation delay of 100ms and a Drop Tail queue of 25 packets at the bottleneck. Simulation settings are assumed to be default (as that mentioned in IV unless specifically specified.

A. Different Bottleneck Bandwidth

Figures 2 (a), (b) and (c) plot the loss rates, throughput and timeouts respectively, for a setup of 30 flows as a function of randomization interval. The buffer size is limited to one-fourth the delay bandwidth product. It can be seen that as the randomization interval increases to 1, the loss rates and the timeouts reduce, while the throughput increases or remains almost the same. Figures 3 (a), (b) and (c) show similar plots for losses, throughput and timeouts for a set of 50 competing flows. Again, it can be inferred from Figure 3 that a randomization interval value of 1 is the best for almost all cases.

B. Different Buffer Sizes

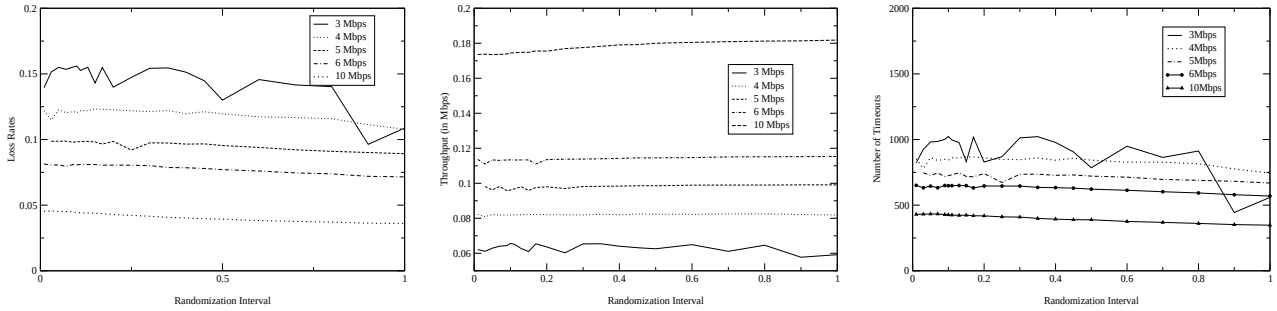
In order to evaluate the effect of buffer sizes, we vary the buffer size at the bottleneck from one-fourth of bandwidth delay product to one bandwidth delay product. The bottleneck link is of 4 Mbps and the end-to-end propagation delay is 100ms. Thus we vary the buffer size from 25 packets (one-fourth bandwidth delay product) to 100 packets (one bandwidth delay product). Again, we plot the losses, throughput and timeouts for 30 flows as a function of randomization interval. Figure 4 show the effect of buffer size. From the Figure 4 it can be inferred that a randomization interval value of 1 gives us the best results vis-a-vis throughput, loss rate and the number of timeouts.

C. Different RTT

In this simulation setup every flow had a unique RTT in the range 80ms to 120ms. The RTT for the $i^{th}, i \in (0, \ldots, N-1)$ flow was $80 + i * (120 - 80)/N$ where N is the total number of competing flows. In Figure 5 we plot the losses, throughput and timeouts for 30 and 50 flows as a function of randomization interval. From the Figure 5 we can conclude that a randomization interval value of 1 suits almost all the simulation metrics.

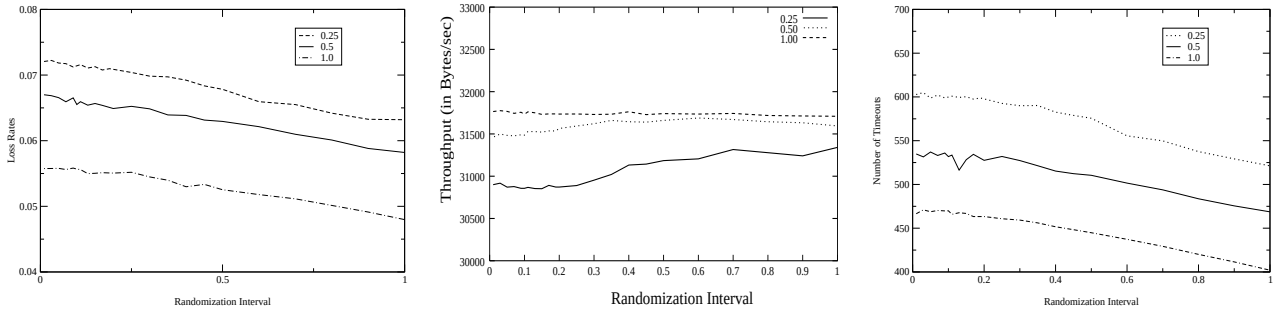
From the above simulations it is evident that a higher value of randomization interval results in increased throughput and lower losses and timeouts. Randomization interval of 1 implies that inter-packet time intervals can lie anywhere between 0 and $2RTT/cwnd$. This means that packets are randomized the most and this results in increased scope for breaking the synchronization, thereby resulting in better performance.

This choice of randomization interval can be intuitively



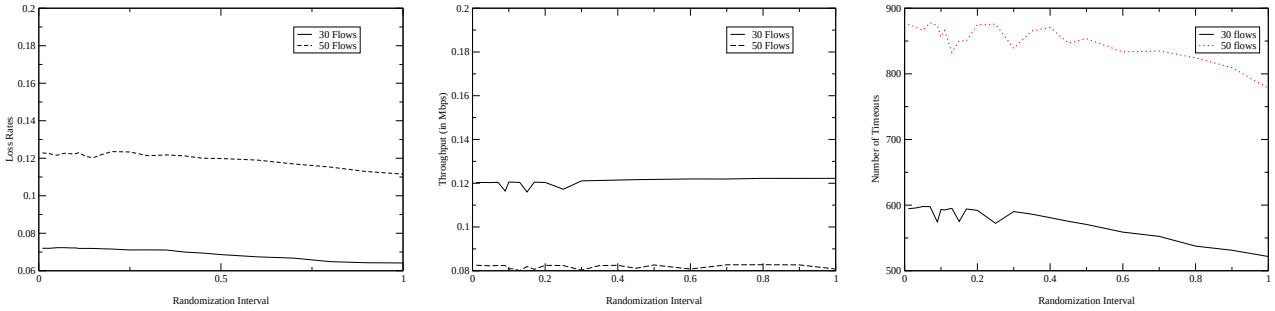
(a) Loss rate (b) Throughput (c) Timeouts

Fig. 3. Loss Rate, Throughput and Timeouts for 50 flows as a function of randomization interval, for different values of bottleneck bandwidth.



(a) Loss rate (b) Throughput (c) Timeouts

Fig. 4. Loss Rate, Throughput and Timeouts for 30 flows as a function of randomization interval, for different values of bottleneck buffer size.



(a) Loss rate (b) Throughput (c) Timeouts

Fig. 5. Loss Rate, Throughput and Timeouts for 30, 50 flows as a function of randomization interval, for varying RTT. The RTT varies from 80ms-120ms, the bottleneck bandwidth is 4Mbps and the buffer size is 25 packets.

explained as following. With a randomized interval value of 1, randomized TCP keeps moving forth between TCP Reno and TCP Paced. (This is because, since randomization interval is Uniform on [-1,1] therefore when the randomized value is -1 then the packets are sent immediately after receiving an ACK akin to TCP Reno. Since the randomization interval is centered around 0, on an average Randomized TCP behaves as Paced TCP.) Intuitively, this entails an early detection of congestion (when the TCP behaves as Reno) and an even distribution of losses and throughput (when TCP behaves as Paced). Thus Randomized TCP takes the best of both Reno and Paced TCP and ensures lesser drops (because of early congestion detection) and fairer throughput.

VI. Throughput, Loss, Timeouts, Fairness and Latency

In this section we compare the performance of Randomized TCP with TCP Reno and Paced TCP. We evaluate all these three schemes for both Bulk data transfers and small Web like transfers. Specifically, we compare the following metrics: average throughput, loss rate, timeouts for bulk data transfers and latency for small web like transfers. We also assess the interaction of Randomized TCP and TCP Reno on a single bottleneck for the metrics throughput, loss rate and timeouts.

A. Bulk Data Transfer

A.1 Same RTT

Figure 6 plots the throughput, loss rate, number of timeouts and fairness for Reno, Paced and Randomized TCP. Though Reno, Paced and Randomized TCP have the same throughput the losses are more for Paced. This is because in slow start, Pacing delays the congestion signal and hence looses a larger number of packets. As the number of flows increase Randomized TCP tends to do the best of the lot.

A.2 Different RTT

To study the performance of Randomized TCP with different RTT values for flows, we varied the RTT of each flow. The RTT of flows were in the range of 80ms to 120ms. The RTT for the $i^{th}, i \in (0, \ldots, N-1)$ flow was $80 + i * (120 - 80)/N$ where *N* is the total number of competing flows. Figure 7 shows the throughput, fairness, loss rates and timeouts as the number of flows are increased from 10 to 50. Randomized TCP is the most fair and also the throughput achieved is marginally higher. However, it is interesting to note that Pacing also achieves almost the same performance as Randomized TCP. TCP Reno maintains its bias against flows with longer RTT (TCP throughput is inversely proportional to the RTT), which is shown by the fairness graph. Because of this bias, Reno's fairness curve is lowest. In [1], the authors contend that bias of TCP against longer RTT flows is considerably reduced with RED gateways due to uniform distribution of losses over time. The similarity of our simulation result to this indicates that randomization succeeds in distributing losses over time (to a certain extent), thereby decreasing TCP's bias towards long flows.

B. Short Web Like Transfers

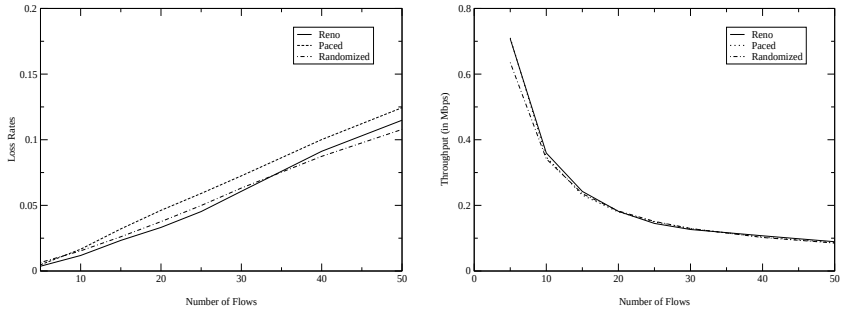
In this section we present the performance of Randomized TCP for short flows. This is more representative of Web transfers. In this simulation we used a bottleneck link of 4Mbps with a round-trip time of 0.1 seconds. The buffer was fixed at one fourth the delay-bandwidth product. 25 flows were always maintained in the network. As soon as any flow finishes, a new flow initiates transfers. We varied the workload from 10 packets to 2500 packets.

Figure 8 (a and b) plots the latencies for Reno, Paced and Randomized TCP. For very short flows, i.e. for a workload of 10 packets to 200 packets, TCP Reno performs the best while Paced TCP performs the worst. Randomized TCP's performance though better than Paced TCP is not as good as Reno's. This can be attributed to the randomness which has been introduced in pacing intervals. Because of this randomization, Randomized TCP breaks ties and achieves better performance than Paced. Reno however, sends packets in bursts and is able to complete most of the transfers in the slow start. For workloads greater than 200 packets, Reno still performs the best, though the difference in the latencies for Reno and Randomized reduce as the workload increases. For Pacing, new flows starting in the slow start saturate the network. Due to late congestion signals in Pacing, many flows, even those who are in congestion avoidance, simultaneously drop packets thus severely diminishing Paced TCP's performance [2]. Reno performs better because Reno flows send packets in clusters, a burst from a particular flow in slow start has only local effect; it does not effect all flows [2].

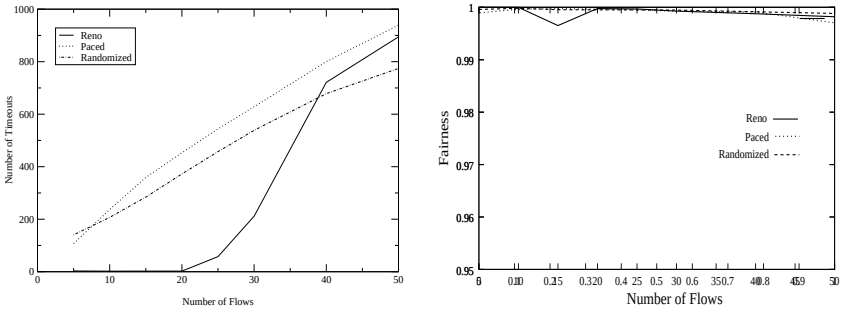
C. Interaction of Randomized TCP with TCP Reno

In this section we will look at the effects of multiplexing TCP Reno and Randomized TCP on the same link. In [2], the authors show that Paced TCP gets beaten down by TCP Reno, when multiplexed on the same link. This is because a single paced connection is more likely to have at least one of its packets encounter severe congestion when multiplexed with a bursty connection [2]. This problem is the same as a source's packets getting synchronized with the buffer overflow event. Hence that flow faces a disproportionate number of losses and a lower throughput [6]. This effect is reproduced in our simulations as shown in Table I where the throughput is considerably lesser for Paced TCP (351.86 Kbps) as against TCP Reno (480.21 Kbps). The end-to-end propagation delay for this experiment was 100ms, the bottleneck link's capacity was 1 Mbps and it was configured with Drop Tail queue with 25 packets of buffer.

However, when Randomized TCP is multiplexed with TCP Reno, the fairness improves considerably. This is seen in Table II where the throughput for the Randomized TCP is 408.92 Kbps when compared to 389.31 Kbps for TCP Reno. This is primarily due to two reasons. Firstly, by

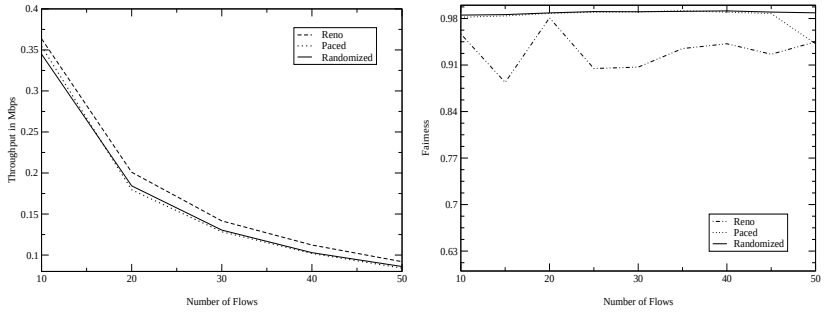


(a) Loss rate (b) Throughput

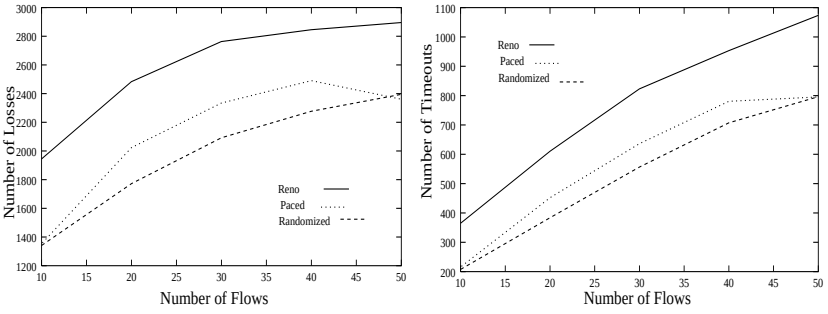


(c) Timeouts (d) Fairness

Fig. 6. Loss Rate, Throughput, Timeouts and fairness with Bulk Data transfer , each flow having same RTT.



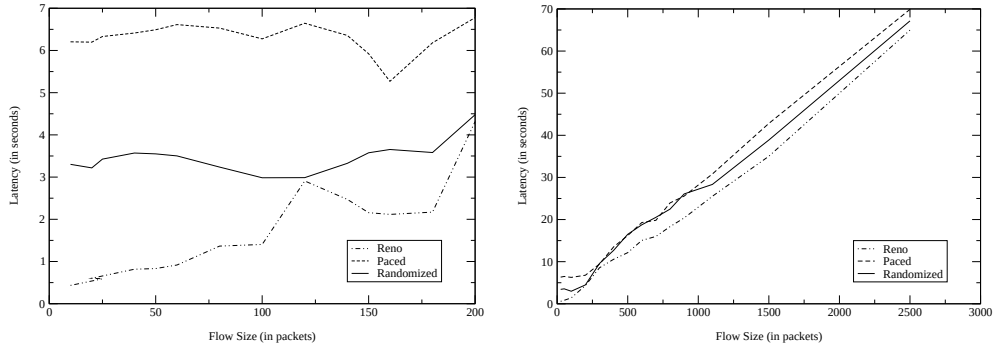
(a) Throughput (b) Fairness



(c) Loss Rates (d) Timeouts

Fig. 7. Throughput, Fairness, Loss Rates and Timeouts for a set of flows where each flow has a different RTT.

modifying the increase parameter α of Randomized TCP we account for the extra delay being introduced by randomization. Secondly, by reduction of synchronization of the source to buffer overflow events, we ensure equitable distribution of drops.



(a) Short Workloads (b) Moderate Workloads

Fig. 8. Latencies for Reno, Paced and Randomized for short and moderate Web like Workloads

TCP Type	Throughput	Losses	Timeouts
Reno	480.21	118	6
Paced	351.86	202	28

TABLE I
COMPARISON OF THROUGHPUT (IN PKTS/SEC), LOSSES AND TIMEOUTS FOR TCP RENO AND PACED.

TCP Type	Throughput	Losses	Timeouts
Reno	389.31	162	22
Random	408.92	210	32

TABLE II
COMPARISON OF THROUGHPUT (IN PKTS/SEC), LOSSES AND TIMEOUTS FOR TCP RENO AND RANDOM.

D. Summary

To summarize the observations of this section:

- For bulk data transfer Randomized TCP performs as well as or better than TCP Reno and Paced TCP in almost all scenarios.
- Specifically, for bulk data transfer with same RTT amongst different flows, with higher multiplexing (of flows) Randomized TCP performs the best by increasing the throughput and fairness, reducing losses and timeouts.
- For bulk data transfers where flows have different RTT, Randomized TCP clearly out-performs TCP Reno and Paced TCP. This is important because this is more representative of the Internet.
- In the scenario where all flows have different RTT and a Drop Tail queue at the bottleneck, randomization reduces the TCP bias against longer RTT flows and achieves a performance similar to RED gateways as mentioned in [1].
- With short web like transfers, Reno out performs Randomized TCP. However as the workloads start to increase Randomized TCP catches up with TCP Reno.
- Randomized TCP and TCP Reno can compete fairly at a bottleneck. This is primarily because of the modification of the increase parameter, α, of the congestion window growth in Randomized TCP. However, Paced TCP loses out to TCP Reno as already shown in [2].

VII. PHASE EFFECTS, SYNCHRONIZATION AND BURST LOSSES

A. Reduction of Phase effects

A.1 Single Bottleneck

In [6] the authors show that phase effects with drop-tail queues can cause a source's loss events to get synchronized with the full queues. Consequently it loses a large number of packets and gets a very low throughput. The authors also note that an appropriate randomization included in the delay would reduce the phase effects. We performed simulations with two flows, one shorter RTT source(60 ms) and another longer RTT source (80 ms) and for differing link capacities to demonstrate the phase effects. We varied the bottleneck capacity but kept the buffer size constant at 25 packets with Drop Tail queues.

If we assume that both flows see the same drop rate then the throughput for the two flows would be distributed as inversely proportional to the RTT (Throughput $\propto$ 1/RTT) [13]. Thus here the throughput should be distributed as 8/14 (0.57) and 6/14 (0.43) of the bottleneck capacity, amongst the 60ms and 80ms sources respectively. Now consider the case when the bottleneck bandwidth is 2 Mbps and both the longer as well as the shorter flow use TCP Reno. The throughputs for the longer and the shorter flow in this case are 119.81 Kbps and 298.93 Kbps respectively. The share of the bottleneck for the two flows is 0.29 (long flow) and 0.71 (short flow) as against the theoretical values of 0.43 (6/14) and 0.57 (8/14) respectively. Therefore, we find that when both the sources use TCP Reno,

phase effects exist as expected. However, with the same 2 Mbps bottleneck, if we randomize one source (in this case, the shorter source), we find that phase effects are considerably reduced as seen in Table III. In fact the throughput for the 80ms and 60ms flows are 166.11 Kbps and 196.3 Kbps respectively. Also their share of the bottleneck are 0.46 (long flow) and 0.54 (short flow) as against the theoretical values of 0.43 (6/14) and 0.57 (8/14) respectively.

A similar statement about the phase effects can be made for the other case where the bottleneck is of 3 Mbps. There too when both the flows use TCP Reno the bottleneck is shared as 0.34 (208/(408.42+208)) for the long flow and 0.64 (408.4/(408.4+208)) for the short flow instead of 0.43 and 0.57 respectively. But when one of flows uses Randomized TCP while the other uses TCP Reno, the bottleneck is shared as 0.44 (241.6/241.6+300) for the long flow and 0.56 for the shorter flow. These two examples elicits that the phase effects are present with TCP Reno and are removed with Randomized TCP.

We investigated another simulation setup with a bottleneck of 1 Mbps, a Drop-Tail queue of 25 packets and 10 flows. In this experiment we had 5 sources each with RTTs of 60ms and 80ms. We first show the occurrence of phase effects when all these sources used TCP Reno, and then we show the removal of phase effects when all these sources used Randomized TCP. But more interestingly, we demonstrate a reduction in phase effects even when *any one* source uses Randomized TCP and the rest use TCP Reno. This implies that a presence of even a *single* Randomized TCP at a bottleneck might be helpful in reducing the phase effects. Thus even an incremental deployment of Randomized TCPs would benefit the entire group of users. The results of this simulation are tabulated in Table IV.

A.2 Multiple Bottleneck

In this section we evaluate the performance of TCP Reno and Randomized TCP with a multiple bottleneck topology. The topology is shown in Figure 9 consists of two bottleneck links of capacity 1 Mbps and delay of 20ms. All the other links in the figure have a capacity of 4 Mbps and delays as shown in Figure 9. The long flows have end-to-end propagation delay of 120ms while the short flows have an end-to-end propagation delay of 60ms. Our simulation setup consist of 2 long flows (they go over both the bottleneck links) and they are generated by S1 and S2, their destinations are D1 and D2, (figure 9) respectively. The two small flows are generated by S3 and S4, with destinations D3 and D4 respectively and they traverse only one bottleneck. We investigate this topology when (S1,S2) and (S3,S4) use TCP Reno and Randomized TCP. Table V tabulates the results for different simulation setups.

Capacity: 2Mbps

RTT	Type	Throughput pkts/sec	Losses	Time-outs
Long	Reno	119.81	684	176
Short	Reno	298.93	581	34
Long	Reno	166.11	320	52
Short	Random	196.3	353	43

Capacity: 3Mbps

RTT	Type	Throughput pkts/sec	Losses	Time-outs
Long	Reno	208.05	558	64
Short	Reno	408.42	251	28
Long	Reno	241.64	253	43
Short	Random	300.08	316	29

TABLE III
PHASE EFFECTS: DISTRIBUTION OF THROUGHPUT IN PROPORTION OF RTTS WITH RANDOMIZED TCP SHOWS REDUCTION OF PHASE EFFECTS IN CONTRAST TO TCP RENO.

We can see from the Table V that there exists phase effects when all the flows use TCP Reno (displayed by the considerable difference in their throughputs) and is subsequently removed when all the flows use Randomized TCP. However, an interesting observation again is that when the short flows use Randomized TCP while the long flows use TCP Reno, we see reduction in phase effects. *This further supports our argument that a presence of even a single randomized flow* **at every** *bottleneck is sufficient to reduce the phase effects and thus achieve a better fairness amongst flows.* In another simulation setup where the long flows use Randomized TCP and the short flows use TCP Reno, we see that the phase effects persists. This is intuitively true too. The long flows are the only sources of potential randomness at the bottleneck, which is visible at the first bottleneck. However, at the second bottleneck the streams arrive in phase because the randomness at the first bottleneck is broken by the "departure process" of the queue. Thus at the second bottleneck there is no randomization to break the phase effects. Hence the long flows get beaten down and the phase effects persist.

B. Synchronization

B.1 Synchronization in Bulk Data Transfer

We ran separate simulations with 2, 3, 10 and 25 flows of Reno, Paced and Randomized TCP and calculated pairwise (between flows) covariance coefficients of congestion windows. The bottleneck bandwidth was 4Mbps and all

RTT	5 Short Reno 5 Long Reno	5 Short Random 5 Long Random	5 Short Reno 4 Long Reno + 1 Long Random	5 Long Reno 4 Short Reno + 1 Short Random
Short	62.6	41.50	45.51	50.02
Long	33.35	33.60	35.80	34.71

TABLE IV

COMPARISON OF THROUGHPUT (IN KBPS) FOR DIFFERENT CONFIGURATION OF COMPETING 5 LONG FLOWS (RTT=80MS) AND 5 SHORT FLOWS (RTT=60MS)

Source	(S1,S2): Reno (S3,S4): Reno	(S1,S2): Random (S3,S4): Random	(S1,S2): Reno (S3,S4): Random	(S1:S2) Random (S3,S4) Reno
S1	50.28	61.28	56.12	50.04
S2	44.08	55.64	59.24	55.00
S3	106.18	83.84	85.20	94.50
S4	108.16	84.36	86.00	98.92

TABLE V

COMPARISON OF THROUGHPUT (IN KBPS) FOR MULTI-BOTTLENECK TOPOLOGY

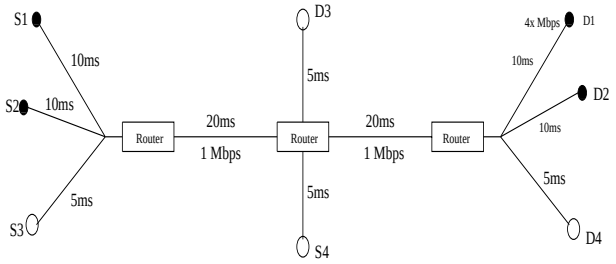


Fig. 9. Multi Bottleneck Topology used in the simulation.

Bandwidth	Reno	Paced	Randomized
3 Mbps	0.4254	-0.4124	0.1721
4 Mbps	0.3152	-0.1839	0.1604
5 Mbps	0.6700	-0.3302	0.0799

TABLE VI

COMPARISON OF COVARIANCE COEFFICIENT OF CONGESTION WINDOW FOR TWO FLOWS FOR TCP RENO, PACED AND RANDOMIZED.

Flow Pair	Reno	Paced	Randomized
(1,2)	0.5183	-0.1454	0.2525
(1,3)	0.5416	-0.1537	0.1422
(1,4)	0.3492	-0.1833	0.1535

TABLE VII

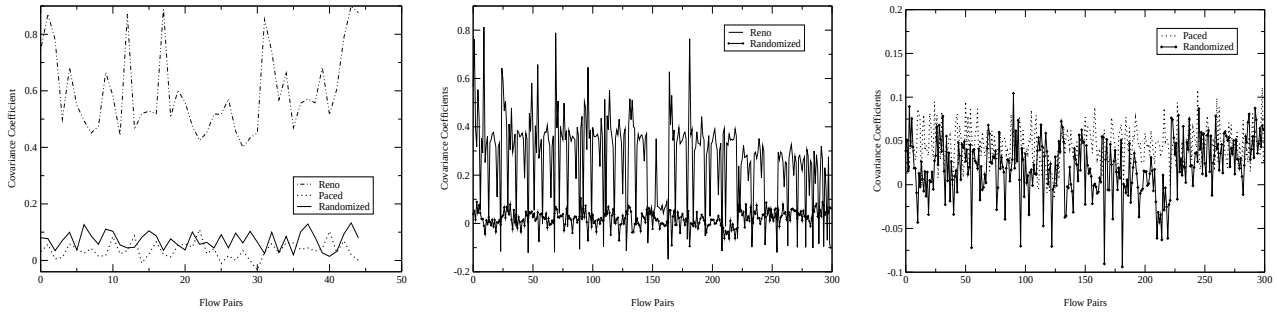
COMPARISON OF COVARIANCE COEFFICIENT OF CONGESTION WINDOWS FOR THREE FLOWS FOR TCP RENO, PACED AND RANDOMIZED.

the flows had a same round-trip time of 0.1 seconds. The simulation was run for 1000 seconds and the congestion window for each flow was sampled using a sample interval of 0.1 seconds, i.e., the congestion window was sampled approximately once every RTT. This sample set was then used to calculate the pairwise covariance coefficients. The buffer was kept at one fourth the bandwidth delay product.

In our first simulation with 2 flows, we varied the bottleneck bandwidth from 3 Mbps to 5 Mbps while keeping the buffer fixed at 25 packets. Table VI shows the covariance coefficients for each of the flows. It can be inferred that the synchronization in Reno increases as the bottleneck bandwidth increases. However Randomized TCP keeps the synchronization low while Paced TCP is out of phase synchronized. Also, it is interesting to note that while the synchronization increases in Reno with increase in bottleneck bandwidth, it decreases in Randomized. Negative values of covariance coefficient show that Paced TCP is out of phase synchronized.

In our second simulation with 3 flows, we kept the bottleneck bandwidth constant. Covariance coefficient values are tabulated in the table VII. Again, it is evident that Reno is the most synchronized and Paced TCP is out of phase synchronized. Also, it can be seen that both Paced and Randomized TCP lead to reduction in the synchronization.

Figures 10(a) and 10(b and c) plot the pairwise covariance coefficients for 10 and 25 flows. The y axis of the graph plots the covariance coefficient against the pair of flows on x axis, i.e., each unit of x axis corresponds to a pair of flows, starting in the order (1,2), (1,3), . . . , (2,3) . . . . Since the graphs for 25 flows are not visible on one graph



(a) 10 flows (b) 25 flows (c) 25 flows

Fig. 10. Covariance coefficients of Congestion Window for (a) Reno, Paced and Randomized (b) Reno and Randomized, (c) Paced and Randomized

we plot it in two. Fig 10(b) plots the covariance for Reno and Randomized TCP and 10(c) plots it for Randomized TCP and Paced TCP. Both Paced TCP and Randomized TCP break synchronization while Reno is highly synchronized. Also, as the number of flows start increasing, Randomized TCP starts to get better than Paced TCP.

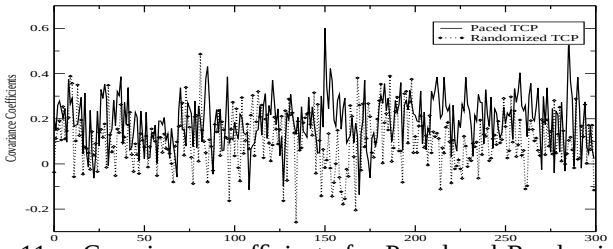


Fig. 11. Covariance coefficients for Paced and Randomized TCP for a transfer of 2500 packets.

B.2 Synchronization with Short Web Transfers

In [2] the authors contend that one of the reasons for higher latency with Paced TCP in short web like transfers is that connections seem to get synchronized. In this simulation setup we have evaluated and verified their arguments. (Simulation setup for this section is identical to what discusses in Section VI-B.) Connections in Paced TCP do get synchronized thereby requiring more time to finish the transfers for some flows. This increases the average latency and hence Paced TCP performs poorly with respect to TCP Reno. However, we find that Randomized TCP is able to break the synchronization and thus achieve lower average latency than Paced TCP.

Figure 11 plots the covariance coefficients of congestion windows for Paced and Randomized TCP. A closer look at the figure shows that the covariance for Randomized TCP is consistently lower than that for Paced TCP. In Paced TCP packets reach the bottleneck at an uniform rate with near perfect interleaving. This causes all sources to lose packets, thereby resulting in all the sources cutting down their windows together, and hence higher covariance. But with randomization, the rate is not uniform at the bottleneck and packets from flows are dropped after differing times due to the extra delay incurred because of randomization. This means that sources decrease their windows at different times and hence the periods of increase and decrease are not as synchronized as in paced, resulting in a decreased covariance coefficient between the flows.

This explains the lower covariance coefficient values for Randomized as compared to Paced TCP.

C. Burst Losses

In this section we investigate the proposition that Randomized TCP reduces the burst losses and also that the drops with Randomized TCP and Drop Tail queues are independent. For testing the first proposition, we varied the bottleneck bandwidth from 1 Mbps to 2 Mbps and the number of sources from 20 to 30. The end-to-end propagation delay was 200ms, the bottleneck buffer was set as 25 packets. We assumed that there is no reverse path congestion and the maximum number of back-to-back packets or burst at the bottleneck will be just 2. We in fact verified the argument of burst loss being equal to 2, by cross checking it with the congestion window trace file for each flow at the bottleneck.

Table VIII shows the results average number of burst losses for TCP Reno and Randomized TCP as the bottleneck bandwidth and the flow multiplexing is increased. It can be inferred from the table that as the number of flows increase, with the bandwidth kept constant, the number of back-to-back losses increase in TCP Reno and decrease (or remain constant) in Randomized TCP. This supports our argument that Randomized TCP reduces burst losses.

It can also be conjectured here that Randomized TCP distributes the loss uniformly over time. This is because, TCP Reno and Randomized TCP have the same congestion control policy the total number of drops are likely to be same for both. Thus, by reducing the burst losses Randomized TCP does distribute losses uniformly over time. This argument is further supported by the evidence

No. of Flows	1 Mbps		2 Mbps	
	Reno	RTCP	Reno	RTCP
20	87	23	1	27
25	119	18	100	31
30	141	15	168	28

TABLE VIII
COMPARISON OF AVERAGE NUMBER OF BURST LOSSES IN RENO AND RANDOMIZED TCP (RTCP).

in Section VI-A.2. There it was shown that Randomized TCP is successful in removing the TCP bias against longer RTT flows with Drop Tail queues. In [1] the authors show that TCP bias against long flows can be reduced by Active Queue Management which distributes losses uniformly over time, specifically RED. The similarity of our simulation results in VI-A.2 indicates that Randomized TCP does succeed in distributing losses uniformly over time.

D. Summary

The observations of this section can be summarized as:
- Phase effects, which persist with TCP Reno with Drop Tail queues, are reduced if Randomized TCP is used.
- Presence of a "single" Randomized TCP flow at every bottleneck can reduce the phase effects at that bottleneck.
- With bulk data transfers randomization reduces the synchronization of windows (thus loss events) as against TCP Reno. This should reduce the queue oscillations.
- Randomized TCP also reduces synchronization with short web-transfers and hence has lower average latency than Paced TCP.
- Randomized TCP drastically reduces the number of burst losses. Specifically its performance increases as the number of flows increase.
- Randomized TCP is also tries to distribute losses uniformly over time with Drop Tail queues at the bottleneck.

VIII. BINOMIAL CONGESTION CONTROL ALGORITHMS

In [3] the authors propose a class of non-linear TCP compatible congestion control schemes called Binomial Congestion Control Schemes (BCCS) for audio and video applications. AIMD, can be considered as one of congestion control schemes in the subset of TCP Compatible BCCS. Formally, the Binomial Congestion Control scheme can be defined as:

$$W_{t+R} \leftarrow W_t + \alpha/W_t^k \quad if \ no \ loss \tag{9}$$

$$W_{t+\delta t} \leftarrow W_t - \beta W_t^l \quad if \ loss \tag{10}$$

where *k* and *l* are window scaling factors for increase and decrease respectively and α and β are increase the decrease proportionality constants. For any given values of α and β TCP Compatible BCCS can be defined by *k+l = 1 : k* $\geq$ *0, l* $\geq$ *0*. Inverse Increase Additive Decrease or IIAD is one such BCCS with *k=1, l=0*. Similarly Square Root Increase and Square Root Decrease or SQRT is defined as *k=0.5, l=0.5*.

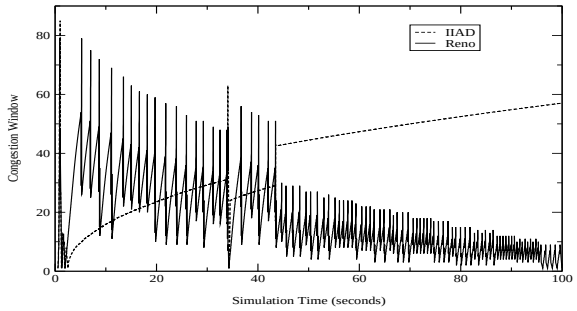
In [3], the authors show that these algorithms, specifically IIAD and SQRT, beat down TCP when sharing a drop-tail gateway and hence suggest the use of RED gateways to maintain fairness. This unfairness is due to unequal distribution of drops amongst these flows. This behavior is seen in Figure 12 a) and c). When we incorporate randomization into binomial schemes as well and make it compete against randomized TCP, we see a marked improvement in fairness as in Figure 12 b) and d), due to the by now familiar reasons of de-synchronization and more uniform distribution of losses. The end-to-end propagation delay for this experiment was 100ms, the bottleneck link's capacity was 1 Mbps and it was configured with Drop Tail queue with 25 packets of buffer.

A. Phase Effects

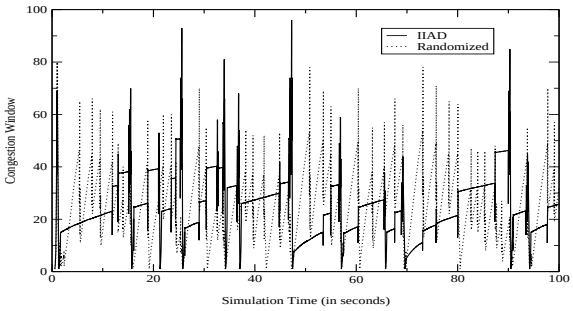
In this section we show the presence of phase effects in Drop Tail Gateways with TCP Reno as first shown in [6]. We use the same simulation setup as disucsses by the authors in [6].

Figure 13 shows the setup for a single bottleneck topology for a 100 ms simulation. The buffer length at the bottleneck gateway is 15 packets which is slightly more than one bandwidth delay product (13 packets) and the packet size used in the simulation was 1000B. In this simulation we vary the RTT of source 1 by varying the delay between source 1 and bottleneck. In figure VIII-A,VIII-A we plot the throughput of Source 2 against the ratio of RTTs of the two sources. As can be seen from the figure VIII-A that for most of the data points, source 2 sees almost no loss while for some particular values of the RTT ratios (between 1.85-2.05) it sees most of the losses showing the presence of phase effects. However, we see that the phase effects are removed if Randomized TCP is used (as shown in figure VIII-A) and the source 2 never sees disproportionately higher percentage of network losses. Figures VIII-A and VIII-A plot the percentage share of the bottleneck of Source 2 for Reno and Randomized TCP respectivelty. Each point in these figures corresponds to the average throughput for the last 50 seconds of the simulation.

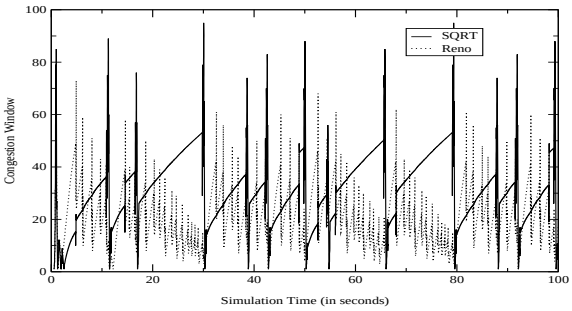
We also evaluated Randomized TCP's performance vis-a-vis phase effects for a multiple bottleneck topology as shown in figure 15. In this simulation we varied the RTT of source 1 by varying the delay between Source 1 and



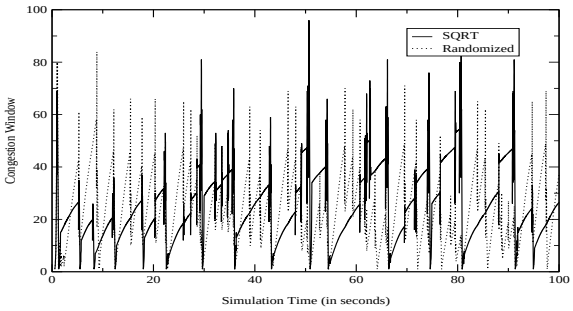
(a) IIAD with Reno



(b) IIAD with Random

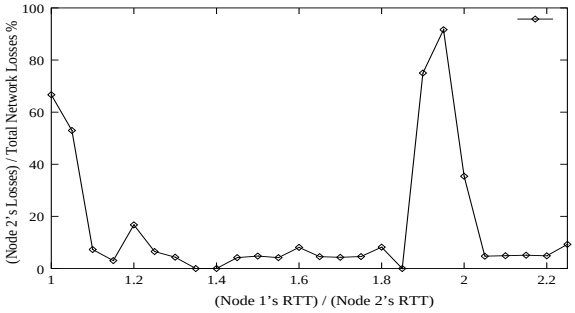


(c) SQRT with Reno

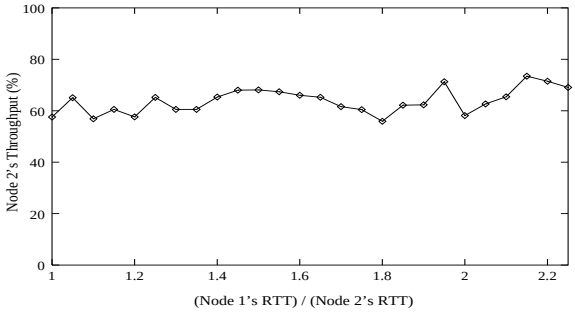


(d) SQRT with Random

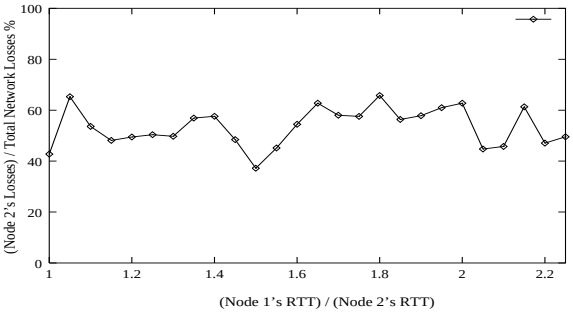
Fig. 12. Performance of Binomial Congestion Control Algorithms with Randomization



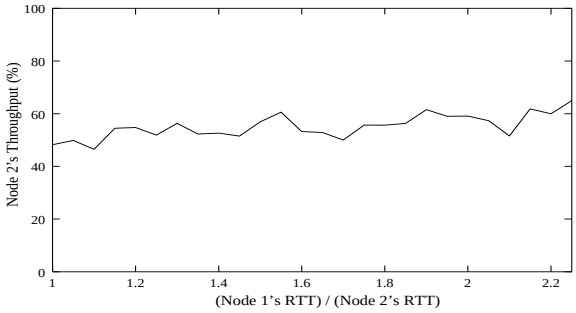
(a) Source 2's % Losses with Reno and Drop Tail



(b) Source 2's % Throughput with Reno and Drop Tail



(c) Source 2's % Losses with Randomized TCP and Drop Tail



(d) Source 2's % Throughput with Randomized TCP and Drop T

Fig. 14. Phase Effects in a Single Bottleneck Topology

bottleneck 1. The packet size used for the simulation was 1000B, the buffer length at each bottlenck was 15 packets (slightly more than 1 bandwidth delay product) and the simulation time was 100 ms. In Figures VIII-A and VIII-A we plot the percentage losses (of the total losses at the second bottleneck) as seen by Source 3 against the RTT ratios of source 1 and 2. Again it can be seen that Source 3 sees almost 80% losses with TCP Reno while the losses are considerably reduced (to about 40%) when Randomized TCP is used. This further verfies the presence of phase effects in Reno and Drop Tail gateways and removal of phase effects with the use of Randomized TCP. Figures VIII-A and VIII-A plot the percentage share of the bottleneck 2 of Source 3 for Reno and Randomized TCP

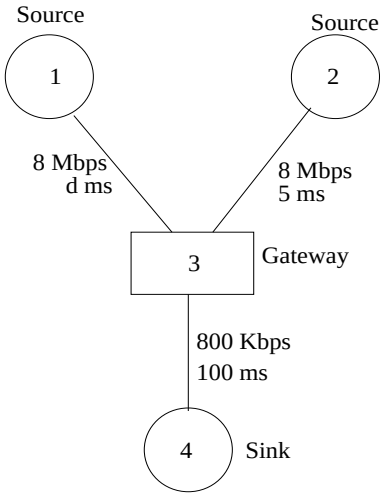


Fig. 13. Single bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.

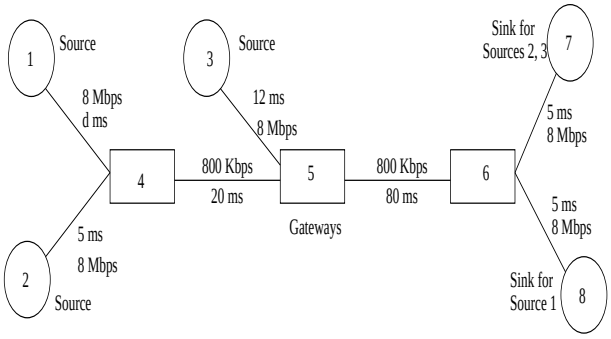


Fig. 15. Multiple bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.

respectivelty. Each point in these figures corresponds to the average throughput for the last 50 seconds of the simulation.

IX. Implementation on the Linux Kernel

We have implemented the Randomized TCP in Linux [16]. The following components were required to implement Randomized TCP 1) a microsecond resolution timer for Linux, 2) a random number generator and 3) a packet scheduling methodology to schedule packets in future. We used UTIME [17] extension to the Linux kernel to introduce microsecond resolution. Scheduling of packets is done on the expiry of this microsecond timer. The Linux kernel provides a random number generator that returns a requested number of random bytes to the module invoked within the kernel. However, our requirement needs the random number to be generated on the byte boundaries but rather or bit boundaries. We wrote functions to create such a random number and also to create both positive and negative random numbers. We tested the implementation with the simple dumb-bell topology. The results seem to conform to the simulation results. The reader is referred to [16] for more information on the implementation.

X. Conclusions and Future Work

In this paper we presented a methodology to introduce randomness in networks by proposing a modification to TCP, called Randomized TCP. In this scheme, we space successive packet transmissions with a time interval $\Delta = RTT(1+x)/cwnd$, where x is a zero mean random number drawn from an Uniform distribution. We showed that Randomized TCP reduces the synchronization and phase effects prevalent with current implementations of TCP as well as Paced TCP. Multiplexing of Randomized TCP with TCP Reno helps in reducing synchronization and phase effects while increasing fairness. Consequently, it has high incentives for deployment.

For a set of connections with different RTTs it was shown that Randomized TCP outperforms Paced TCP and TCP Reno considering fairness, throughput, drop rates and timeouts. Randomized TCP also reduces the bias against connections with larger RTTs with Drop Tail queues. The presence of a single Randomized flow at a bottleneck is sufficient to reduce the phase effects thereby motivating incremental deployment. Randomized TCP can also distribute losses uniformly over time emulating RED like properties. Additionally, when Randomized TCP is extended to Binomial congestion control schemes, there is a remarkable improvement in fairness, when competing with Reno.

References

[1] A. A. Abouzeid and S. Roy , "Analytic Understanding of RED Gateways with Multiple Competing TCP Flows", *Proceedings of IEEE GLOBECOM*, November 2000.

[2] A. Aggarwal, S. Savage, and T. Anderson, "Understanding the performance of TCP pacing," *Proceedings of IEEE INFOCOM*, pp. 1157-1165, Tel-Aviv, Israel, March 2000.

[3] D. Bansal and H. Balakrishnan, "Binomial Congestion Control Algorithms", *Proc. IEEE INFOCOM*, Anchorage, AK, April 2001.

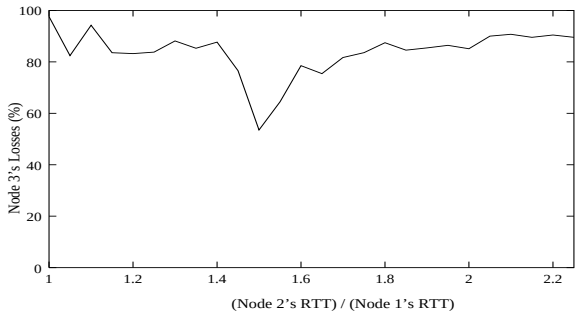
[4] D-M. Chiu and R. Jain, "Analysis of increase and decrease algorithms for congestion avoidance in computer networks," *Computer Networks and ISDN Systems*, vol. 17, no. 1, pp. 1-14, June 1989.

[5] S. Floyd, "Connections with multiple congested gateways in packet-switched networks Part 1: One-way traffic," *Computer Communication Review*, vol.21, no.5, pp. 30-47, Oct 1991.

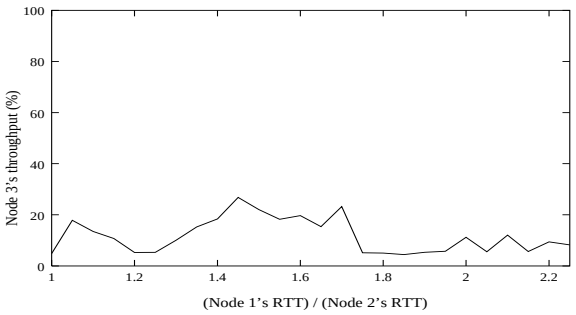
[6] S. Floyd and V. Jacobson, "On traffic phase effects in packet-switched gateways," *Internetworking: Research and Experience*, vol. 3, no. 3, pp. 115-156, September 1992.

[7] S. Floyd and V. Jacobson, "Random early detection gateways for TCP congestion avoidance," *IEEE/ACM Transactions on Networking* vol. 1, no. 4, pp. 397-413, August 1993.

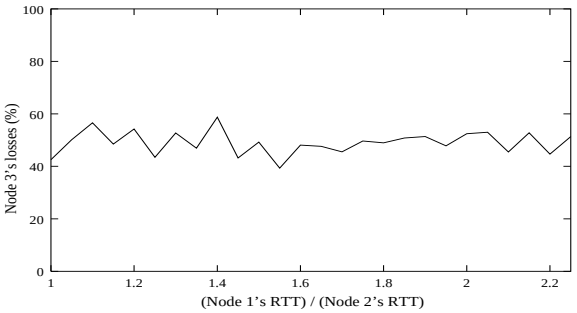
[8] E. Hashem, "Analysis of random drop for gateway congestion control," *Report LCS TR-465*, p. 103, Laboratory for Computer Science, MIT, Cambridge, MA, 1989.



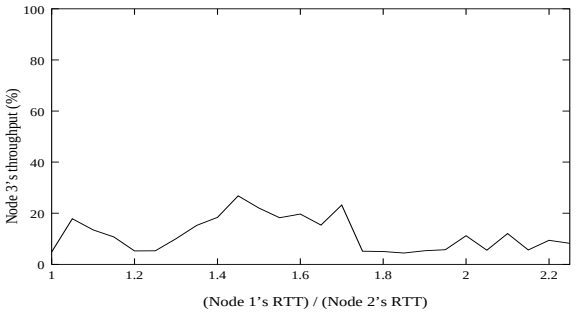
(a) Source 2's % Losses with Reno and Drop Tail



(b) Source 2's % Throughput with Reno and Drop Tail



(c) Source 2's % Losses with Randomized TCP and Drop Tail



(d) Source 2's % Throughput with Randomized TCP and Drop T

Fig. 16. Phase Effects in a Multiple Bottleneck Topology

[9] J. Ke and C. Williamson, "Towards a Rate Based TCP Protocol for the Web", *Proc. of MASCOTS*, San Francisco, CA, August 2000.

[10] M. May, J. Bolot, C. Diot and B. Lyles, "Reasons not to deploy RED," *Proc. of IWQoS*, pp. 260-262, London, UK, June 1999.

[11] M. May, T. Bonald and J.-C. Bolot, "Analytic evaluation of RED performance," *Proceedings of IEEE INFOCOM*, pp. 1415-1424, Tel-Aviv, Israel, March 2000.

[12] J. Mogul, "Observing TCP dynamics in real networks," *Proc. of ACM SIGCOMM*, pp. 305-317, Baltimore, MD, August 1992.

[13] J. Padhye, V. Firoiu, D. Towsley and J. Kurose, "Modeling TCP Reno performance: A simple model and its empirical validation," *IEEE/ACM Trans. on Networking*, vol. 8, no. 2, pp. 133-145, April 2000.

[14] S. Shenker, L. Zhang and D. Clark, "Some observations on the dynamics of a congestion control algorithm," *ACM Computer Communications Review*, vol 20, no. 4, pp. 30-39, October 1990.

[15] L. Zhang, S. Shenker, and D. Clark, "Observations on the dynamics of a congestion control algorithm: The effects of two-way traffic," *Proceedings of ACM SIGCOMM*, pp. 133-147, Zurich, Switzerland, September 1991.

[16] M. Mehta, "On Randomizing the Sending Times in TCP: An Implementation", Masters Thesis, ECSE, R.P.I., Dec 2001.

[17] "UTIME: Microsecond Resolution Timers for Linux", http://www.ittc.ku.edu/utime/

[18] N. Plotkin and P. Varaiya, "The entropy of traffic streams in ATM virtual circuits," *Proceedings of IEEE INFOCOM*, pp. 1038-1045, June 1994.