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

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 through end-to-end congestion control protocols. For the TCP case we call it *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. Randomized TCP, by introducing randomization in the network, reduces synchronization, phase effects and bias against bursty traffic and longer RTT flows, prevalent with current implementations of TCP and Drop Tail Gateways. Our results suggest that by randomizing the sending times we can successfully emulate almost all the beneficial features of RED except congestion avoidance. Moreover, these benefits of randomization can be achieved even when it is partially deployed. We also introduce randomization in Binomial schemes and show performance improvements with Drop Tail queues.**

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 (or correspondingly 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], [11]. 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 $RTT(1+x)/cwnd$, 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 thus is very attractive from an implementation perspective.

The increased randomization increases the entropy of the system which correspondingly reduces the queue sizes thereby improving the stability of the system [17]. Our results show that Randomized TCP reduces phase effects and synchronization even when multiplexed with TCP Reno flows. Also it substantially reduces burst losses and removes the bias against longer RTT flows. In addition, *the benefits of randomization can be reaped even when it is partially deployed.* 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 investigate the impact of randomization on a class of slowly varying congestion control schemes called Binomial schemes [3] and 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 parameter tuning of RED. However, we wish to emphasize that unlike RED which is a congestion avoidance scheme, Randomized TCP is just a congestion control scheme. Thus Randomized TCP does not emulate the congestion avoidance features of RED, at best it provides the other beneficial features of RED which were achieved by introducing randomization in the network (by dropping packets probabilistically).

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

- It proposes a distributed (end system based) 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
- Synchronization of windows which persist with TCP Reno and Drop Tail queues are reduced
- The presence of a single Randomized TCP flow can remove the bias against longer RTT flows at the bottleneck thereby motivating incremental deployment
- Randomized TCP, instead of burst losses, distributes losses over time, emulating 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 VI evaluates the bias against longer RTT flows, phase effects, synchronization and burst losses for Randomized TCP and TCP Reno. Finally we present the conclusions and future work in Section IX.

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 Paced TCP removes synchronization with long flows, 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 is beaten by TCP Reno when competing together.

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, phase-effects, bias against long RTT flows etc 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.

The main purpose of this paper is not to compare the performance of Randomized TCP with Paced TCP [2], [9] but to illustrate the benefits of introducing randomization in the network. However for the purpose of completeness we have also compared the performance of Randomized TCP and Paced TCP (Section VII). The main focus of this paper is to illustrate how by introducing randomization into the network (by randomizing the sending times) we can remove almost all the deficiencies of Tail Drop gateways, specifically bias against bursty and longer RTT flows, synchronization and phase-effects.

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. The Randomized TCP's sending time algorithm is stated in Section III-C.

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. 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 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. Using a simple M/M/1/K queueing analysis, similar to that in [11], in Section III-B we try to get a quantitative feel of the probability of a packet getting dropped with Randomized TCP.

Randomizing the sending times also 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 (in Section III-A) 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

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

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

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

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

$$\sum_{i=1}^{w} x_i \sim N(0,\sigma^2), \quad \sigma^2 = w * \frac{I^2}{3} \quad (4)$$

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

$$E[\sum_{i=1}^{w} x_i \mid (\sum_{i=1}^{w} x_i \geq 0)] = \sqrt{\frac{2wI^2}{3\pi}} \quad (5)$$

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

$$\overline{RTT}_{eff} = R + \frac{1}{2w}(\sqrt{\frac{2wI^2}{3\pi}} + \frac{I}{2}) \quad (6)$$

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.

B. Queueing Analysis to show reduction in Burst losses with Randomized TCP

Consider a *M/M/1/K* queueing system where the packets arrive according to a batch Poisson process; specifically, bursts (or batches) of B packets arrive according to a Poisson process of rate λ. Further, let us denote by $\pi(k)$ as the stationary distribution of *k* number of packets in the queue. Then using the PASTA (Poisson Arrival See Time Averages) property the probability of a packet drop in a Tail Drop router with TCP as input can be calculated as:

$$P_{TD} = \pi(K) + \pi(K-1)\frac{B-1}{B} + ... + \pi(K-B+1)\frac{1}{B}$$

Using the same model we will now calculate the probability of a packet being dropped for Randomized TCP. We first note that the size of burst, B, will now be changed because Randomized TCP paces the packets. Hence we first try to find the new burst size (given that the original burst size was B) and then calculate the packet drop probability. Figure 1 shows the epochs at which the packets are sent. Let us call the time instants at which the packets from a Paced TCP would have been sent as *centered epoch*. These *centered epochs* now represent the time instants around which we randomize the sending times of packets in Randomized TCP. Suppose a packet is sent at some time, *x* after the centered epoch (as shown in figure 1). Let us also define the length of the packet as *L* bits and the bottleneck link capacity as *C* bits/sec. Further, let the window size at steady state be *W* (B $\leq$ W) and let *RTT* denote the round-trip time. Then the probability, *p*, of packets from a burst of *B*, arriving back-to-back at the bottleneck router can be calculated as

$$p = \int_0^{\frac{RTT}{W}} \left(\frac{1}{2}\frac{x}{\frac{RTT}{W}}\right) \left(\frac{1}{2}\frac{\frac{L}{C}}{\frac{RTT}{W}}\right) dx = \frac{1}{8}\frac{L}{C} \quad (7)$$

Note that the now, $B^{'} = min(B * \frac{L}{8C}, B)$, represents the

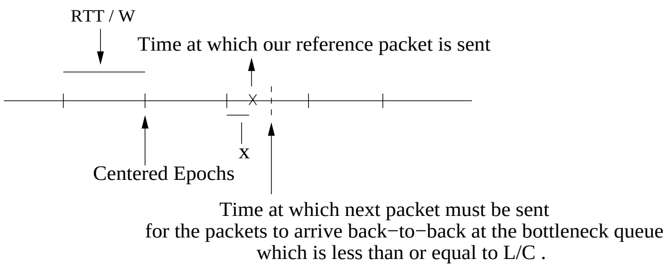

RTT / W
Time at which our reference packet is sent
x
x
Centered Epochs
Time at which next packet must be sent
for the packets to arrive back−to−back at the bottleneck queue
which is less than or equal to L/C .

Fig. 1. Packet Sent Times with Randomized TCP

upper bound on the number of back-to-back packets which can be received at a bottleneck with Randomized TCP and a burst of size B. Also note that the above analysis holds true iff $\frac{L}{8C} \leq \frac{RTT}{W}$ which holds true for WANs and MANs.

Using the above equation, the probability that a packet gets dropped with Randomized TCP and drop tail router can be calculated as

$$P_{TDR} = \pi(K) + \pi(K-1)\frac{B^{'}-1}{B^{'}} + ... + \pi(K-B^{'}+1)\frac{1}{B^{'}} \; .$$

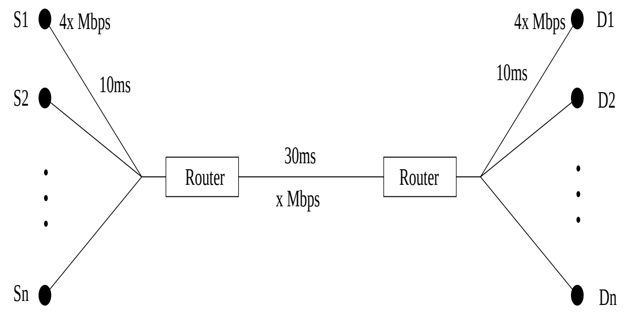


Fig. 2. Topology used in the simulation.

Thus from the above observation we can conclude that the probability that a packet gets dropped with Randomized TCP and drop tail queue decreases. However, it should be noted that Poisson arrivals do not capture the exact packet arrivals in the Internet. Nevertheless, this exercise is just intended to show that the probability of burst losses are reduced with Randomized TCP and have been validated by our simulation results in Section VI-D.

C. Randomized TCP Pseudo-code

Define by α the original increase parameter for the TCP Reno and by *R* the RTT. Then the Randomized TCP's algorithm can be stated as

```
• Send a packet. Schedule the next packet to be sent at time $t = \frac{RTT}{cwnd}(1+x)$ where x is Uniformly distributed on [-1,1].
• Let $t^{'}$ be the arrival time of next ack. Then
  – If $t^{'} < t$ send the next packet at t.
  – Else send the next packet after $\frac{RTT}{cwnd}y$ where y is uniformly distributed on [0,1].
• At each RTT (estimate) update recalculate the new increase parameter as $\alpha_{new} = \alpha\left(\frac{RTT_{new}}{R}\right)^2$
```

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 recovery 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 2 shows the topology used in the simulations. The access links were configured at a rate 4 times greater than that of the bottleneck link and all the links use Drop Tail queues. 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.

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

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

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 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 the randomization interval I in section V. Using these simulations we propose a value of the interval for optimal performance.

Section VI shows the performance of Randomized TCP with respect to phase effects, synchronization amongst flows and burst losses. In Section VII-A, we investigate the interaction between Randomized TCP and Reno, for fairness, throughput, losses and timeouts. We also present the result of comparative performance of Randomized, Paced and Reno TCP in Section VII for the following set of metrics: throughput, losses, timeouts, fairness and latency for both bulk-data transfer and short-web like transfers. Finally in Section VIII we present the results of extension of Randomization to Binomial schemes.

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 present 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. Due to space constraints we present the summary of the findings of these simulations here, the detailed results (and simulation settings) can be found in the Technical Report [16].

The results obtained from the simulation indicate that a higher value of randomization interval results in increased throughput and lower losses and timeouts. Hence we choose the randomization interval to be 1. Randomization interval of 1 implies that inter-packet time intervals can lie anywhere between 0 and $2RTT/cwnd$. This means that packets are randomized over a larger interval and this results in increased scope for breaking the synchronization, thereby resulting in better performance.

VI. Bias Against Long Flows, Phase Effects, Synchronization and Burst Losses

A. Bias Against Long Flows

It has been widely reported that Drop Tail gateways have a bias against long flows [7]. In this section we first demonstrate this bias and its reduction with the use of Randomized TCP. We present the results with single as well as multiple bottleneck topologies.

A.1 Single Bottleneck

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 bias against long flows. We varied the bottleneck capacity but kept the buffer size constant at 25 packets with Drop Tail queues. The simulation time was 100 seconds.

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 and 0.57 respectively. Therefore, we find that when both the sources use TCP Reno, bias against longer flow exist as expected. However, with the same 2 Mbps bottleneck, if we randomize one source (in this case, the shorter source), we find that bias against longer flow is considerably reduced as seen in Table I. 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 and 0.57 respectively. This beneficial effect of Randomized TCP is preserved even if we randomize the longer flow.

A similar statement about the bias against longer flow 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 for the long flow and 0.64 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 for the long flow and 0.56 for the shorter flow. These two examples elicits that the bias against longer flows 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 bias against longer flows when all these sources used TCP Reno, and then we show the removal of this bias when all these sources used Randomized TCP. But more interestingly, *we demonstrate a reduction in bias 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 bias against flows with larger RTT. Thus even an incremental deployment of Randomized TCPs would benefit the entire group of users*. The results of this simulation are tabulated in Table II.

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 3 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 3. 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 denoted by (S1-D1) and (S2-D2) source-destination pairs and two small flows denoted by (S3-D3) and (S4-D4) source- destination pairs, as shown in figure 3. We investigate this topology when (S1,S2) and (S3,S4) use TCP Reno and Randomized TCP. Table III tabulates the results for different simulation setups.

We can see from the Table III that there exists bias against flow(s) with longer RTT when all the flows use

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

Capacity: 2Mbps

RTT	Type	Throughput pkts/sec	Losses (%)	Time-outs
Long	Reno	119.81	5.7	176
Short	Reno	298.93	1.1	34
Long	Reno	166.11	1.9	52
Short	Random	196.3	1.8	43

Capacity: 3Mbps

RTT	Type	Throughput pkts/sec	Losses (%)	Time-outs
Long	Reno	208.05	2.7	64
Short	Reno	408.42	0.6	28
Long	Reno	241.64	1.2	43
Short	Random	300.08	1.05	29

TABLE I
BIAS AGAINST LONGER FLOWS: DISTRIBUTION OF THROUGHPUT IN PROPORTION OF RTTS WITH RANDOMIZED TCP SHOWS REDUCTION OF BIAS IN CONTRAST TO TCP RENO.

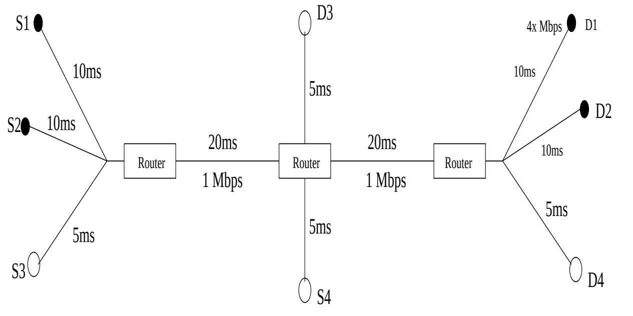


Fig. 3. Multi Bottleneck Topology used in the simulation.

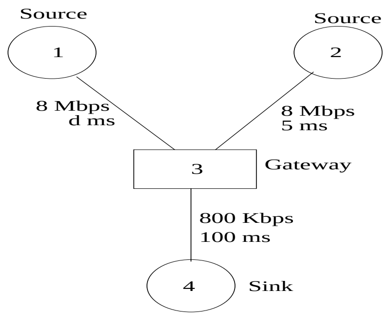


Fig. 4. Single bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.

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 this bias. *This further supports our argument that a presence of even a single randomized flow* **at every** *bottleneck is sufficient to reduce the bias against longer flow(s) 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 bias 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 bias against longer flows. Hence the long flows get beaten down and the bias persists.

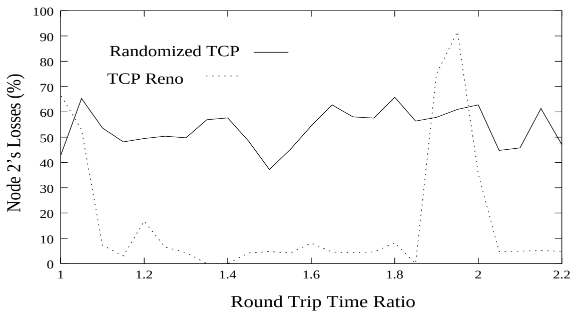
B. *Phase Effects*

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. In this section we show the presence of phase effects in Drop Tail Gateways with TCP Reno as first shown in [6]. Subsequently, using the same simulation setup we show reduction in phase-effects with the use of Randomized TCP. We use the same simulation setup as discusses by the authors in [6]. Since phase-effects can be shown by either disproportionately high number of losses or low throughput in this paper we chose losses to demonstrate phase-effects. Each point in these losses-time plot corresponds to the average losses for the last 50 seconds of the simulation.

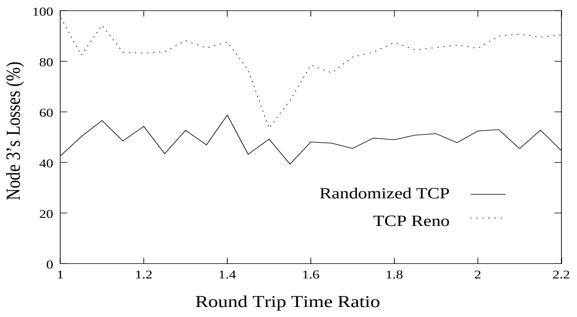
Figure 4 shows the setup for a single bottleneck topology for a 100 ms simulation, a bottleneck buffer of 15 packets and the packet size of 1000B. In this simulation

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



(a) Single Bottleneck: Node 2 does not see disproportionate losses with Randomized TCP, Phase effects reduced.



(b) Multiple Bottleneck: Node 3 does not see disproportionate losses with Randomized TCP, Phase effects reduced.

Fig. 5. Phase Effects

we vary the RTT of source 1 by varying the delay between source 1 and bottleneck. In figure 5(a) we plot the losses of Source 2 against the ratio of RTTs of the two sources. As can be seen from the figure 5(a) that for most of the data points, source 2 sees almost no loss (source 1 sees all the losses) 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 and the source 2 never sees disproportionately higher percentage of network losses.

We also evaluated Randomized TCP's performance vis-a-vis phase effects for a multiple bottleneck topology as shown in figure 6. In this simulation we varied the RTT of source 1 by varying the delay between Source 1 and bottleneck 1. The packet size used for the simulation was 1000B, the buffer length at each bottleneck was 15 packets (slightly more than 1 bandwidth delay product) and the simulation time was 100 ms. In Figure 5(b) 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 verifies the presence of phase effects in Reno and Drop Tail gateways and removal of phase effects with the use of Randomized TCP.

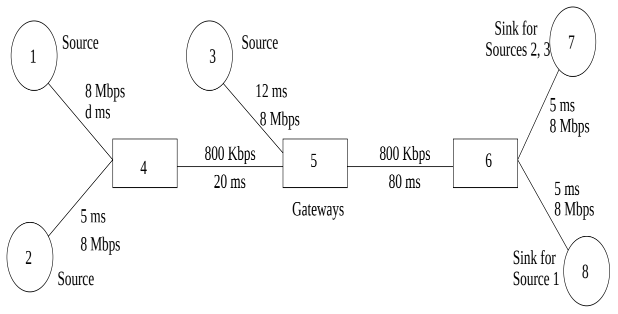


Fig. 6. Multiple bottleneck Simulation Setup to show phase effects with Reno and Drop Tail Gateways.

C. Synchronization

C.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. We maintained the default simulation setup as described in Section IV and the simulation time was 1000 seconds. 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.

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 IV shows the covariance coefficients for each of the flows. It can be inferred that

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 IV
COMPARISON OF COVARIANCE COEFFICIENT OF CONGESTION WINDOW FOR TWO FLOWS FOR TCP RENO, PACED AND RANDOMIZED. (VALUE AROUND 0 IS GOOD.)

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 V
COMPARISON OF COVARIANCE COEFFICIENT OF CONGESTION WINDOWS FOR 3 FLOWS FOR TCP RENO, PACED AND RANDOMIZED. (VALUE AROUND 0 IS GOOD.)

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.

In our second simulation with 3 flows, we kept the bottleneck bandwidth constant. Covariance coefficient values are tabulated in the table V. 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 7(a) and 7(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 we plot it in two. Fig 7(b) plots the covariance for Reno and Randomized TCP and 7(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.

C.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. For the simulation we used a bottleneck link of 4Mbps, a RTT of 100 ms and a buffer of 25 packets. 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.

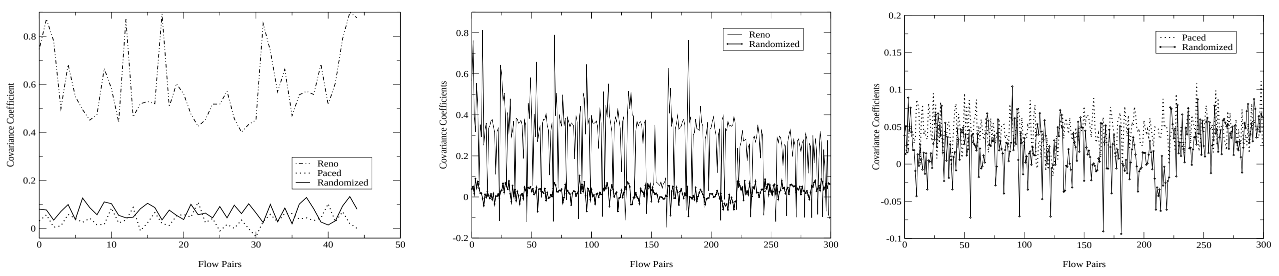
Figure 8 plots the covariance coefficients of congestion windows for Paced and Randomized TCP. A closer look shows that the covariance for Randomized TCP is *consistently* lower than that for Paced TCP. In Paced TCP packets reach the bottleneck at an uniform rate with near perfect interleaving. This 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.

D. 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 1-2 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 also verified this argument by cross checking the burst loss size with the congestion window trace file for each flow at the bottleneck.

Table VI 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 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 makes the losses distributed. This argument is further supported by the results in Section VI-A. 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



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

Fig. 7. Covar. coeff. of Congestion Window for (a) Reno, Paced & Randomized (b) Reno & Randomized, (c) Paced & Randomized

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

TABLE VI

Comparison of average number of burst losses in

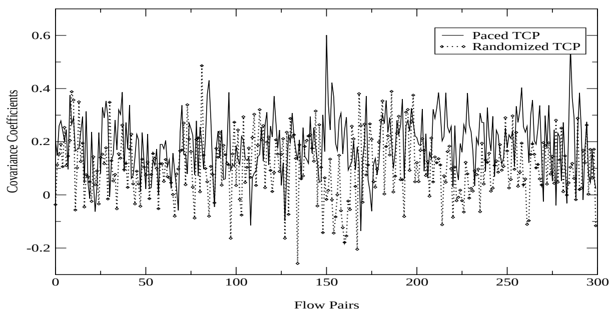


Fig. 8. Covariance coefficients for Paced and Randomized TCP for a transfer of 2500 packets. (Value around 0 is good.)

can be reduced by Active Queue Management which distributes losses uniformly over time, specifically RED. The similarity of our simulation results in VI-A suggest that Randomized TCP does succeed in making losses independent by distributing them over time.

E. Summary

The observations of this section can be summarized as:

• Randomized TCP increases the fairness amongst competing flows of different RTTs by removing the bias against the longer RTT flows (as found with TCP Reno) with Drop Tail queues.

• Presence of a "single" Randomized TCP flow at every bottleneck (Drop Tail Gateways) can reduce the bias against longer RTT flow at that bottleneck.

• Phase effects, which persist with TCP Reno with Drop Tail queues, are reduced if Randomized TCP is used.

• 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 reduces synchronization with short web-transfers. This should lower average latency.

• Randomized TCP drastically reduces the number of burst losses. Specifically its performance increases as the number of flows increase.

• With Drop Tail queues Randomized TCP tries to distribute losses over time thus making them appear independent.

VII. Throughput, Loss, Timeouts, Fairness and Latency

In this section we report the performance of Randomized TCP as compared to TCP Reno and Paced TCP. We evaluate all these three schemes for both Bulk data transfers (with all the flows having same RTT and in the other case where every flow has a different RTT) and small Web like transfers. Specifically, we compared the following metrics: average throughput, loss rate, timeouts for bulk data transfers and latency for small web like transfers. We varied the workload of the small web-like transfers between 10 to 2500 packets. Due to reasons of space constraints we only summarize our results here. Randomized TCP performed as well as or better than Paced and Reno TCP for most of these scenarios. The performance was notably better in the case where all flows had different RTTs. Randomized TCP also reduces the latencies of short-web like transfers as compared to those with Paced TCP. However for very small transfers Reno still does the best. Detailed results for these sections can be accessed from the Technical Report [16].

A. Interaction of Randomized TCP with TCP Reno

This section presents the result 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 VII where the throughput is considerably lesser for Paced

TCP Type	Throughput	Losses (%)	Timeouts (%)
Reno	480.21	2.45	0.1
Paced	351.86	5.74	0.8
Reno	389.31	4.2	0.5
Random	408.92	5.1	0.8

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

TCP (351.86 Kbps) as against TCP Reno (480.21 Kbps). The RTT for this experiment was 100ms, the bottleneck link's capacity was 1 Mbps and it was configured with Drop Tail with 25 packets of buffer.

However, when Randomized TCP is multiplexed with TCP Reno, the fairness improves considerably. This is seen in Table VII 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 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.

B. 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 every flow has 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 performs better than Randomized TCP. However as the workloads start to increase Randomized TCP catches up with TCP Reno. Small workload flows complete their transaction in slow-start (or with very small windows). As such, if we randomize the windows when they are small, randomization generally delays the sending times which results in increased latency. Moreover, our re-characterization of increase parameter (Section III-A) does not come into play because it works for congestion avoidance phase. As such, we conjecture that one should not randomize the sending times when the windows are small (less than 4) and during the slow-start. However, these inferences are at best intuitive and need to be evaluated in detail. One could also calculate the adjustment factor for slow start (just like we did for steady state). We leave this as future work.
• 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].

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. Formally, the Binomial Congestion Control scheme can be defined as:

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

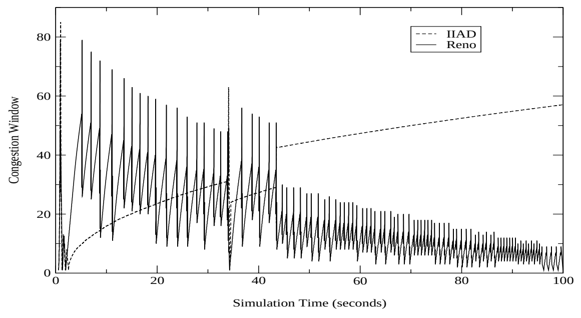
$$W_{t+\delta t} \leftarrow W_t - \beta W_t^l \quad if\ loss \tag{10}$$

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*. We refer the reader to [3] for a more detailed description of Binomial congestion control schemes.

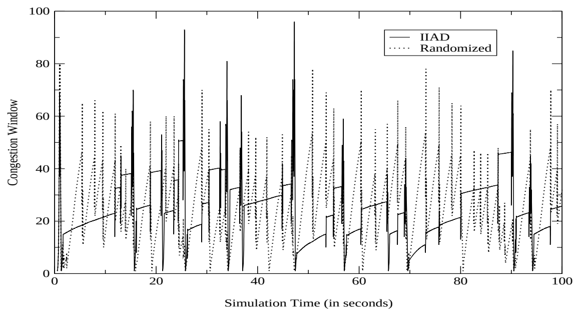
In [3], the authors show that these algorithms, specifically IIAD and SQRT, beat down TCP when sharing a drop-tail gateway and hence suggest the use of RED gateways to maintain fairness. This unfairness is due to unequal distribution of drops amongst these flows. This behavior, for IIAD, is seen in Figure 9 a). 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 9 b) due to a more uniform distribution of losses. The RTT 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. Due to space constraints we do not present the results for SQRT, which are however available in the Technical Report [16].

IX. CONCLUSIONS

In this paper we presented a methodology to introduce randomness in networks through end-to-end congestion



(a) IIAD with Reno



(b) IIAD with Random

Fig. 9. Performance of Binomial Congestion Control Algorithms with Randomization

control schemes. For the TCP case, we call it 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, by introducing randomization in the network, reduces synchronization, phase effects and bias against bursty traffic, prevalent with current implementations of TCP and Drop Tail Gateways. We have also analytically characterized the new increase parameter for Randomized TCP to make it compete fairly with TCP. This was necessary because randomizing the sending times increases the RTT and as such the Randomized TCP losses to TCP Reno.

Randomized TCP 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 bias against longer RTT flows thereby motivating incremental deployment. Randomized TCP also reduces the burst losses and can also distribute losses over time thus emulating RED like properties. Multiplexing of Randomized TCP with TCP Reno helps in reducing synchronization and phase effects while increasing fairness. Additionally, when Randomized TCP is extended to Binomial congestion control schemes, there is a remarkable improvement in fairness, when competing with Reno. Consequently, it has high incentives for deployment.

Finally our results indicate that, *Randomized TCP can emulate the beneficial effects of RED in a distributed manner* without the complexities and unfavorable aspects of parameter tuning of RED. In addition, the benefits of randomization can be reaped even when it is partially deployed. However, we wish to emphasize that unlike RED which is a congestion avoidance scheme, Randomized TCP is just a congestion control scheme. Thus Randomized TCP does not emulate the congestion avoidance features of RED, at best it provides the other beneficial features of RED which were achieved by introducing randomization in the network. We are currently working on implementation of Randomized TCP in the Linux Kernel.

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,* MIT, Cambridge, MA, 1989.

[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,” *Proc. of INFOCOM,* 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 et. al, “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,” *Proc. of ACM SIGCOMM,* , Zurich, Switzerland, Sept. 1991.

[16] K. Chandrayana et. al., “On Randomizing the Sending Times in TCP and other Window Based algorithm”, ECSE-NET-2001-1, http://networks.ecse.rpi.edu/tech_rep.html.

[17] N. Plotkin and P. Varaiya, “The entropy of traffic streams in ATM virtual circuits,” *Proc. of IEEE INFOCOM,* June 1994.