Performance of TCP Congestion Control With Explicit Rate Feedback

Aditya Karnik and Anurag Kumar

***Abstract*—We consider a modification of TCP congestion control in which the congestion window is adapted to explicit bottleneck rate feedback; we call this RATCP (*Rate Adaptive TCP*). Our goal in this paper is to study and compare the performance of RATCP and TCP in various network scenarios with a view to understanding the possibilities and limits of providing better feedback to TCP than just implicit feedback via packet loss. To understand the dynamics of rate feedback and window control, we develop and analyze a model for a long-lived RATCP (and TCP) session that gets a time-varying rate on a bottleneck link. We also conduct experiments on a Linux based test-bed to study issues such as fairness, random losses, and randomly arriving short file transfers. We find that the analysis matches well with the results from the test-bed. For large file transfers, under low background load, ideal fair rate feedback improves the performance of TCP by 15%–20%. For small randomly arriving file transfers, though RATCP performs only slightly better than TCP it reduces losses and variability of throughputs across sessions. RATCP distinguishes between congestion and corruption losses, and ensures fairness for sessions with different round trip times sharing the bottleneck link. We believe that rate feedback mechanisms can be implemented using distributed flow control and recently proposed REM in which case, ECN bit itself can be used to provide the rate feedback.**

***Index Terms*—Congestion control, rate feedback, TCP.**

I. Introduction

TCP WINDOW adaptation is based on *implicit feedbacks* from the network; acknowledgment cause the congestion window to increase, and packet losses (indicated by timeouts or duplicate acknowledgment) cause the window to decrease. Owing to this blind rate adaptation mechanism, TCP has often been found to be inefficient, in terms of underutilization of link capacity and low session throughputs, and unfair in its throughput performance. In view of this, various modifications to basic TCP congestion control algorithms have been proposed and investigated. This includes TCP-Vegas ([1]), TCP-SACK ([2]), New-Reno, limited transmit mechanism, and larger initial windows ([3]). Estimates of the available bandwidth are used to set the slow start threshold in [4]. More recently, in TCP-Westwood ([5]) a mechanism of faster recovery by setting the value of slow start threshold and congestion window after a loss event based on available bandwidth estimates has been introduced. This approach does not require comprehensive changes to the basic TCP implementation. However, the benefits are limited since TCP still has to infer congestion information on the end-to-end basis. TCP-Vegas, for example, uses round-trip-time (RTT) measurements to estimate the actual and expected throughput to set the congestion window.

Much more performance improvement can be expected with explicit participation of the network in the congestion control of TCP. One such mechanism is active queue management based on RED or ECN. RED, unlike tail drop buffers, drops packets randomly (based on average queue length) at a router buffer before it gets full. This forces TCP sources to back off before the congestion takes place. RED is aimed at eliminating loss synchronization and controlling the average queueing delay [6]. However, various studies show that benefits from RED are not clear [7], [8] shows that RED degrades TCP performance under a variety of scenarios. In addition, RED requires several parameters to be configured on each router interface, and if not properly configured it can induce network instability [9]. Lack of a clear understanding of what an optimal average queue size should be and absence of a systematic way of setting parameters means that the choice of parameter is often empirical, even arbitrary.

Since dropping packets is rather a suboptimal way of conveying congestion information to TCP sources, ECN marks packets with 1 bit of congestion notification [10]. Upon receipt of such a congestion indication, TCP reduces its congestion window. However, ECN relies on the underlying queue management mechanism which in most cases is RED (thereby, inheriting the problems of parameter configuration). ECN has been found to reduce packet losses but does not necessarily improve throughputs [11]. Though ECN presents a new way of providing explicit congestion feedback to sources, it requires modifications to routers as well as TCP stack. Moreover, clear understanding of benefits of ECN is still a research issue [12], [13].

Compared to the "coarse" feedback provided by ECN, TCP would clearly benefit from a more "sophisticated" feedback from the network. Our view is particularly motivated from the study of TCP performance over a rate controlled transport like ATM/ABR. TCP is seen to benefit from the underlying rate control even when the two control loops, namely TCP window control and ATM/ABR rate control, do not interact ([14]). It would, therefore, be interesting to study the performance gains (and limits) when a more detailed feedback such as the available rate information is made available to TCP congestion control.

Rate feedback (or rate control) for TCP over ATM or IP is not new. An innovative approach suggested in [15] involves maintaining an acknowledgment bucket at the edge device in an ATM network and releasing the acknowledgment based upon the available rate. An explicit window feedback based on

Manuscript received July 4, 2001; revised December 12, 2003; approved by IEEE/ACM Transactions on Networking Editor T. V. Lakshman.

A. Karnik was with the Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, 560012 India. He is now with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: aditya.karnik@gmail.com).

A. Kumar is with the Department of Electrical Communication Engineering, Indian Institute of Science, Bangalore, 560012 India (e-mail: anurag@ece.iisc.ernet.in).

Digital Object Identifier 10.1109/TNET.2004.842237

the buffer occupancy at the edge device of an internetwork of rate-controlled and nonrate-controlled segments is considered in [16]. A more direct approach is taken in [17] (TCP over ATM) and [18] (TCP over IP). Fair rate is calculated by a rate allocation algorithm and is translated into window information which is fed back as receiver window advertisement. It has been shown that the rate feedback reduces packet losses, and improves fairness and throughputs under various scenarios. Our work differs from previous work in the following respects.

At the conceptual level, our objective is to understand the performance limits of providing better feedback directly to TCP sources than just implicit feedback via packet losses. Even if rate is fed back to the TCP source, there is still an issue of utilizing it efficiently. We suggest changes to TCP's adaptive window algorithm to utilize rate feedback more effectively and call this modification *Rate Adaptive TCP* (RATCP). We assume that the network is somehow able to feedback fair session rates to TCP sources. The TCP sources then adapt their congestion windows based on this rate feedback and an RTT estimate. *Thus our concern in this paper is to study the performance implications of feeding available bottleneck rate information directly into TCP windows, assuming that such rate information can be obtained and that mechanisms exist for feeding it back to the sources.*

The existing studies in this area are simulation and experimentation based. Moreover, our aim is not just to enumerate benefits of rate feedback under various scenarios, but to study the dynamics of rate feedback and window control, for example, the effect of feedback delay, rate mismatches etc. We, therefore, develop an analytical model for obtaining the throughput of a long-lived (or *persistent*) session sharing a bottleneck link with short-lived (or *ephemeral*) sessions that arrive and depart randomly; the ephemeral sessions are assumed to be ideally rate controlled and the persistent session uses RATCP or TCP both without the fast-retransmit feature; thus the persistent session has a time varying fair rate. *The analysis models the round trip delay, the bottleneck buffer, slow start, congestion avoidance and rate feedback.* We proceed by identifying a certain Markov regenerative process, and calculating the TCP throughput as the reward rate in this process. This analysis allows us to characterize the effect of variation of rate feedback on the performance of TCP.

Our experimental setup comprises an implementation of RATCP in Linux; the bottleneck link is emulated in the Linux kernel. This setup and the analysis are cross-checked with each other. The test-bed also provides quantitative results for the other cases, including results for RATCP and TCP with fast-retransmit and recovery. In particular, we compare the performance of RATCP and TCP on our test-bed in the following scenarios:

1) A persistent session over a bottleneck link with random loss.
2) Two persistent sessions with different round-trip times sharing a bottleneck link; *both* the sessions use either RATCP or TCP.
3) Two persistent sessions on a link, one using RATCP and the other TCP.
4) A link being shared by ephemeral sessions that randomly arrive and depart.
5) Scenario (4) on a link with random losses.

Scenarios (1) and (5) are particularly interesting from the point of view of wireless networks. Thus, with the analysis and the experiments we study the effect of rate variations, the comparison with the ideal rate adaptive protocol, and the ability of RATCP to distinguish congestion and random losses.

At the practical level, our results would be useful to the designers of edge management devices where such techniques could be employed. Rate feedback entails putting in place mechanisms to generate and carry the available rate information. An approach, which does not need such mechanisms, is to estimate the available bandwidth at the TCP sources as in TCP-Westwood [5]. However, this does not impose fairness in the network; a source which sends at a rate matching the estimated available rate can easily be starved by a greedy source. A robust network-based mechanism can be implemented by distributed rate control [19], [20]. Whereas in ATM/ABR mechanism to carry rate feedbacks is in place (e.g., RM cells), in IP networks, with some modifications, ECN bit can actually be used to provide the rate feedback to TCP. Toward the end of the paper, we briefly discuss the use of rate estimation algorithms and explicit binary (or multi-bit) feedback schemes as suggestions for implementation of RATCP.

This paper is organized as follows. In Section II, we describe the RATCP algorithm. In Section III, we develop a stochastic model for RATCP and present its analysis; the proofs are presented in Section VIII. The experimental setup is explained in Section IV followed by numerical results in Section V. In Section VI, we discuss rate estimation and feedback schemes which can be used for implementing RATCP. We conclude in Section VII.

II. RATCP: Window Adaptation With Rate Feedback

A. A Naive Rate to Window Translation

Consider a TCP session through a bottleneck link. If the round trip propagation delay for the session is Δ, and the fair share of the bottleneck rate is R, then the congestion window for this session should be $W = R \cdot \Delta + \beta$, where β is a target buffer backlog for this session. Now if the fair rate for the session is time varying ($R(t)$), and $\hat{\Delta}(t)$ is an estimate (at the source) of Δ at t, then a simple, naive rate adapted window would be to take $W(t) = R(t-\Delta) \cdot \hat{\Delta}(t) + \beta$, where $R(t-\Delta)$ is the available rate as known to the source at time t. Note that, $\hat{\Delta}$ measured at the TCP source includes queueing delays. One way to get better estimates is to track the *base RTT*, i.e., the minimum RTT seen by the source. β allows a session to take advantage of the transient rate increments.[1] In this paper, we wish to study how such a naive feedback performs.

B. Window Adaptation

The rate adaptive window adaptation strategy is the following (W^{cong} denotes the congestion window, and is the window actually used for transmission control):

- *Slow start* is carried out either at connection startup, or at the restart after a timeout. We use the rate information for

[1]The importance of this parameter is well demonstrated by our results. TCP-Vegas also uses a similar parameter [1].

setting the slow start parameters: W^{cong} at timeout is set to 1, and the slow start threshold (*ssthresh*) is set to the value of W^{rate} at the timeout epoch. If during slow start $W^{rate} < W^{cong}$ then the congestion window is dropped to W^{rate}, and congestion avoidance is entered. This is appropriate, since it is as if the *ssthresh* has been adjusted downward.

- During congestion avoidance, at time t, we compute $W^{cong}(t+) = \min\{W^{cong}(t), W^{rate}(t)\}$. If the congestion window reduces as a result of $W^{rate}(t) < W^{cong}(t)$, then it means that more than the desirable number of packets are in the network. Acks following such a window reduction do not cause the window to increase until the number of unacknowledged packets corresponds to the new window. This adds a phase of inactivity in the modified TCP. Normal congestion avoidance behavior continues after the number of outstanding packets matches the new congestion window. If during congestion avoidance W^{cong} becomes less than *ssthresh* (due to a W^{rate} feedback) then slow start is not re-initiated. This is reasonable, since it is as if the *ssthresh* has been adjusted downward, and we are now just entering congestion avoidance. This also implies that *ssthresh* no longer differentiates the phases of the TCP algorithm; we need to introduce a separate variable for this purpose.
- If *fast-retransmit and fast-recovery* are implemented then upon receiving K (typically $K = 3$) duplicate acks we set $W^{cong} \leftarrow \min(W^{cong}, W^{rate})$ (instead of $W^{cong} \leftarrow (W^{cong}/2) + K$ as in TCP-Reno), and the missing packet is retransmitted. After every additional acknowledgment received W^{cong} is increased by 1. Upon receipt of the ack for the resent packet, congestion avoidance resumes, as described above.

We call these modified TCP algorithms, *Rate Adaptive TCP (RATCP)*. We will compare RATCP and TCP without fast-retransmit and fast-recovery, and will call these versions RATCP-OldTahoe and TCP-OldTahoe. The versions with fast-retransmit and fast-recovery will be called RATCP-Reno and TCP-Reno.

III. A Model and Its Analysis

Analysis, even if approximate, is essential for providing insight into factors that affect the performance of a protocol. In addition, although simulations and experiments are usually used to validate analysis it is just as important to cross-check simulations and experimental results with analyzes for at least some cases.

We develop an analytical model for the performance of RATCP OldTahoe in the following network scenario. There is a persistent RATCP session, that shares a bottleneck link with other elastic sessions. The elastic sessions are assumed to be *ideally rate controlled* and ephemeral, i.e., they arrive at random epochs, bring a random amount of data to transfer and depart after completing their transfers. When there are m ephemeral sessions, we assume that these sessions use exactly $m/(m+1)$ of the link capacity of C packets/s, and the persistent session's share is $C/(m+1)$ pkts/s. Thus the fair bandwidth available to the persistent session is randomly time varying. Ephemeral sessions should not be likened to the "background traffic"; they are ideally rate-controlled, hence their packets do not occupy the link buffers. Their role is to make the available rate at the bottleneck link time varying; otherwise, after rate feedback takes effect, there will be ideal performance. This amounts to an assumption of per flow queueing and round robin service over the flows at the router.

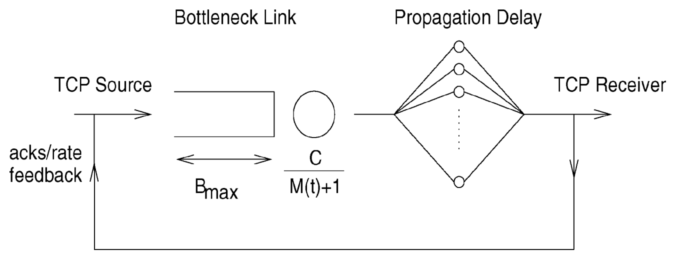


Fig. 1. A queueing model of the persistent TCP session.

Our analysis captures the important effect of time scales of rate variations at the bottleneck link as compared to the propagation delay. Thus, the role of our analysis is to study *the dynamics of rate changes at the bottleneck link, round trip delay and the rate adaptive TCP window control*. Because of these issues, this analysis does not lead to close form expressions; however, the numerical results provide insights into the dynamics we intend to study.[2]

Fig. 1 shows a schematic queueing model of the persistent TCP session. The bottleneck link is modeled as a finite buffer queue with maximum buffer size of $B_{\max}$ packets, and a server with time-varying rate $C/(M(t)+1)$ pkts/s. Note that the ephemeral sessions are only modeled as modulating the rate available to the TCP session, and hence the link buffer only holds packets from the TCP session. Δ denotes the fixed round trip delay and is modeled as an infinite server with fixed service time equal to Δ. We assume that the link from the source to the bottleneck link is infinitely fast.

The continuous time processes for this model are hard to analyze. Instead, we follow the analysis procedure developed in [14]. Define the epochs $t_k = k\Delta$, $k = 0, 1, 2, \ldots$. Observe that none of the packets that are in the delay queue at time t_k will still be in that queue at time t_{k+1}, and any packet that arrives into the delay queue during $(t_k, t_{k+1}]$ will still be there at time t_{k+1}. We thus consider the processes embedded at the epochs $\{t_k, k \geq 0\}$ (see Fig. 2), and define

$$\{Z_k, k \geq 0\} = \{(B_k, D_k, W_k^{cong}, W_k^{rate}, M_k), k \geq 0\}$$

where, at epoch t_k, W_k^{rate} and W_k^{cong} denote the rate window and the congestion window for the persistent RATCP session. M_k denotes the number of ephemeral sessions on the link, B_k the number of packets in the link buffer, and D_k the number of packets in the propagation queue; this is the total number of packets and acks in transit.

A. *Model for the Rate Modulating Process* $\{M_k\}$

We assume that the ephemeral sessions arrive and depart at the discrete epochs t_k. Thus, during the interval $(t_k,$

[2]Previous analytical work which arrived at closed-forms for TCP performance has not dealt with time varying rates.

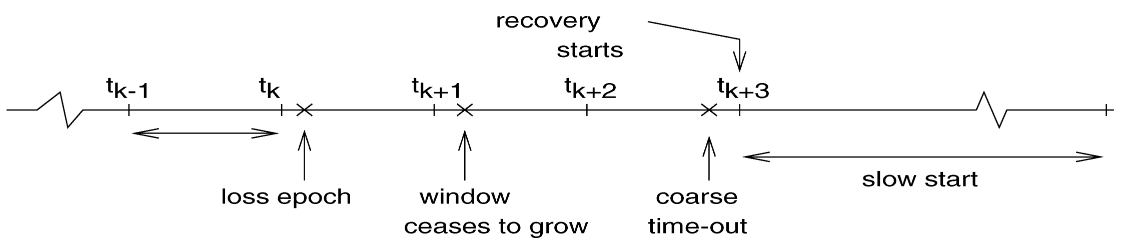


Fig. 2. Evolution of $\{Z_k, k \geq 0\}$, showing the model for timeout based loss recovery.

t_{k+1}) the rate available to the TCP session is constant at $R_k = C/(M_k + 1)$. A new arrival occurs at any t_k with probability λ; i.e., the inter-arrival times are geometrically distributed (taking values that are multiples of Δ). The amount of data to be transferred by an ephemeral session is denoted by L, and is taken to be exponentially distributed with mean μ, in units of $\Delta \cdot C$. When there are n sessions sharing the link, each session is served at C/n. Let p_n be the probability that a session active at t_k and being served at a rate C/n in (t_k, t_{k+1}) departs by t_{k+1}. Then it is clear that

$$p_n = 1 - \exp\left(-\frac{\mu}{n}\right).$$

Thus given that there are m ephemeral sessions active at t_k, each one of them independently completes in the interval (t_k, t_{k+1}) with probability p_{m+1}. It follows that $\{M_k, k \geq 0\}$ is a DTMC. Note that since the TCP session is persistent it is always counted as being active in the per session rate calculation, and hence $\{M_k, k \geq 0\}$ evolves independently of the other components of the process $\{Z_k\}$.

Denote by P_M the transition probability matrix of $\{M_k, k \geq 0\}$. Then we have

$$[P_M]_{ij} = \begin{cases} \lambda, & \text{if } j = i+1 \\ a(i,j), & \text{for } 0 \leq j < i \\ 1 - \lambda - \sum_{j=0}^{i-1} a(i,j), & \text{if } j = i \end{cases}$$

where $a(i,j) = \binom{i}{i-j} (p_{i+1})^{i-j}(1 - p_{i+1})^j$. This also implies that the number of sessions which depart in (t_k, t_{k+1}) conditioned on the number of sessions present at t_k is binomially distributed.

B. Model of Window Adaptation to Rate Feedback

The rate window is calculated from the instantaneous rate information known at the TCP source of the tagged session. We assume that delay Δ is *known* at the TCP source and that it receives a (delayed) rate feedback every round trip time. With these assumptions, our analysis gives a bound on the performance of TCP with rate feedback. In experiments we will study the effect of estimation errors in the base round trip delay. Owing to one Δ delay in rate feedback, the rate window calculated at t_k is given by

$$W_k^{rate} = \lfloor R_{k-1} \cdot \Delta \rfloor + \beta \tag{1}$$

where $R_{k-1} = C/(M_{k-1} + 1)$. Then, the window adaptation policy implies that

$$W_{k+}^{cong} = \min\{W_k^{cong}, W_k^{rate}\}. \tag{2}$$

Note that the assumption of an infinite rate link between the TCP source and the bottleneck link implies that, if W^{cong} is not rate limited (and there are no losses), then $W_k^{cong} = B_k + D_k$ (any window increase immediately results in as many more packets in the network). We can then see how, adaptation to the rate window reduces the packet losses due to buffer overflow, since it controls the backlog of packets in the buffer to a target value of β. This can be seen as follows. Note that $D_k \leq R_{k-1} \cdot \Delta$. Assuming that the bottleneck link was busy throughout (t_{k-1}, t_k) ($\Rightarrow D_k = R_{k-1} \cdot \Delta$), we find from (1) that, $B_k > \beta \Rightarrow B_k + D_k > \beta + D_k \Rightarrow W_k^{cong} > W_k^{rate}$. Hence, $W_{k+}^{cong} = W_k^{rate}$ and then excess packets are drained from the network.

C. Evolution of $\{Z_k\}$, and a Process $\{X_k\}$

We make some basic assumptions in order to make the analysis of the $\{Z_k\}$ process tractable.

- The source immediately transmits new packets as soon as its window allows it to; these arrive instantaneously at the link buffer.
- Packet transmissions from the link do not straddle the epochs $\{t_k\}$.
- During each interval $(t_k, t_{k+1}]$, the acknowledgment (acks) arrive at the TCP source at the rate R_{k-1}.

Let $Z_k = (b, d, w^{cong}, w^{rate}, m)$. Then $W_{k+}^{cong} = \min(w^{cong},\ w^{rate})$. Note that there can be at most d acks during (t_k, t_{k+1}). These acks may trigger new packet arrivals into the link buffer. In congestion avoidance we have the following possibilities.

1) If $W_{k+}^{cong} < b + d$ (this would occur if $w^{rate} < w^{cong}$), then $h = b + d - W_{k+}^{cong}$ packets need to be removed from the network before congestion avoidance resumes. Since the number of acks that will be received in (t_k, t_{k+1}) is d, we first have the following two cases.
   - **Case 1:** $d < h \Rightarrow$ not enough acks are received, the source is inactive throughout (t_k, t_{k+1}) and $W_{k+1}^{cong} = W_{k+}^{cong}$; there is no packet loss.
   - **Case 2:** $d > h \Rightarrow$ congestion avoidance commences during (t_k, t_{k+1}) after the first h acks are received. There may be losses in (t_k, t_{k+1}) after h acks are received.

2) **Case 3:** $W_{k+}^{cong} = b + d \Rightarrow$ congestion avoidance continues; as acks are received, W^{cong} is incremented and new packets are generated. There may be losses in (t_k, t_{k+1}).

If a loss does occur during (t_k, t_{k+1}), adjustments to W_{k+}^{cong} may occur till the ack for the packet just prior to the one that is lost is received (see Fig. 2). We assume that this ack arrives at the source in (t_{k+1}, t_{k+2}). At this point the source starts a coarse timer. We assume that the coarse timeout occurs during (t_{k+2}, t_{k+3}) and the recovery begins at t_{k+3} (see Fig. 2). Recalling that we are not modeling the fast-retransmit procedure, denote by L_{ss}, the duration of the slow start phase (in number of Δ intervals). L_{ss} will vary with each loss instance, but developing an indexing for it would be cumbersome. Then the recovery is over at $t_{k'}, k' = k + 3 + L_{ss}$ and the congestion avoidance phase begins.

Define the embedded epochs $\{T_k\}$ by $T_0 = t_0$, and for $k \geq 0$

$$T_{k+1} = \begin{cases} T_k + \Delta & \text{no loss in } (T_k, T_k + \Delta) \\ T_k + (3 + L_{ss}) \cdot \Delta & \text{loss in } (T_k, T_k + \Delta) \end{cases}$$

where L_{ss} denotes the duration of the slow start phase. Finally, define the embedded process $\{X_k = Z_{T_k}, k \geq 0\}$ with $X_0 = Z_0$.

Proposition 3.1: $\{(X_k, T_k), k \geq 0\}$ is a Markov Renewal process.

The proof of Proposition 3.1 is presented in Section VIII.

D. Computation of Throughput

Given the Markov Renewal Process $\{(X_k, T_k), k \geq 0\}$, a *reward* V_k is associated with the k^{th} cycle (T_k, T_{k+1}), as the number of successful packets accounted in that interval. Let $V(x)$ and $U(x)$ respectively denote the reward and the length of the cycle beginning with $X_k = x$. Denote by $\pi(x)$, the stationary probability distribution of the Markov chain $\{X_k, k \geq 0\}$. Then denoting by γ the throughput, and by E_π the expectation with respect to $\pi(x)$, from the Markov Renewal-Reward Theorem we have

$$\gamma = \frac{E_\pi V}{E_\pi U}. \tag{3}$$

If packet loss does not occur in (t_k, t_{k+1}), then we count the reward as the number of acks received by the source, that is, D_k, denoted by $D(x)$ to show dependence on $X_k = x$. When packet loss does occur, the reward is accounted as the sum of D_k acks that return to the source in (t_k, t_{k+1}), the number of packets *ahead* of the packet that is lost $D_{before\ loss}(x)$, and $D_{slow\ start}(x)$, the number of packets transmitted in the slow start phase. We do not count any of the packets transmitted successfully *after* the lost packet. Thus,

$$V(x) = \begin{cases} D(x) & \text{w.p. } 1 - p_{loss}(x) \\ D(x) & \\ +D_{before\ loss}(x) & \\ +D_{slow\ start}(x) & \text{w.p. } p_{loss}(x) \end{cases}$$

$$U(x) = \begin{cases} \Delta & \text{w.p. } 1 - p_{loss}(x) \\ (3 + L_{ss})\Delta & \text{w.p. } p_{loss}(x). \end{cases}$$

Analysis of TCP without rate control is similar to the analysis described above. Since at the embedded epochs $\{T_k\}$, the equation $W_k^{cong} = B_k + D_k$ holds, we need to consider only the four-dimensional process:

$$\{Z_k, k \geq 0\} = \{(B_k, D_k, M_{k-1}, M_k), k \geq 0\}.$$

M_{k-1} needs to be considered as it determines the rate at which acks return from the delay queue to the source. Additional details of the analysis are provided in [21].

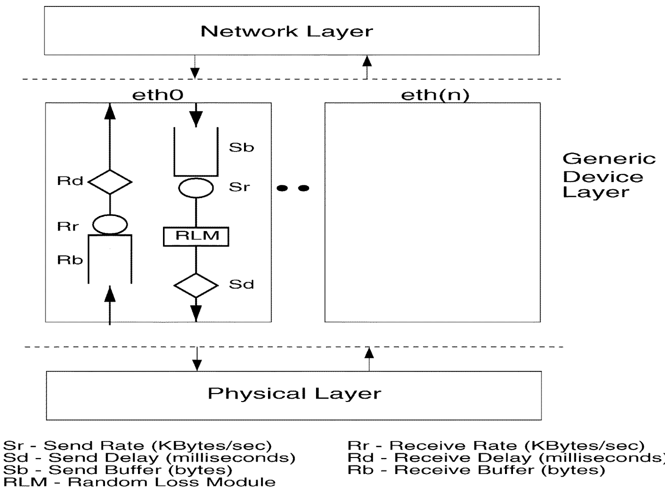

Network Layer
eth0
eth(n)
Sb
Rd
Sr
Rr
RLM
Rb
Sd
Generic Device Layer
Physical Layer
Sr - Send Rate (KBytes/sec)
Sd - Send Delay (milliseconds)
Sb - Send Buffer (bytes)
RLM - Random Loss Module
Rr - Receive Rate (KBytes/sec)
Rd - Receive Delay (milliseconds)
Rb - Receive Buffer (bytes)

Fig. 3. Implementation of the Wide-Area Link Emulator (WALE) in the Linux kernel.

IV. Experimental Setup

The experimental results for the network of Fig. 1 reported here are obtained from a Linux based Wide-Area Link Emulator (WALE) [22]. WALE, as shown in Fig. 3, models a full duplex WAN link on a single Ethernet interface. The link parameters namely, send/receive buffer sizes, send/receive transmission rates, and send/receive propagation delays are emulated at the generic device driver layer. To emulate lossy links, e.g., satellite links, random loss module (RLM) creates packet losses with a user specified probability. All these parameters can be set using a link configuration utility. File transfers are run *using the actual Linux TCP code* modified according to RATCP.

Modifications include new variables for rate window and for phase to differentiate slow start and congestion avoidance. The exact rate feedback with appropriate delay is artificially provided to the TCP sender using a new system call. The rate window is then calculated using this rate feedback and TCP base RTT estimate for every incoming ack (tcp-ack routine); this models the case when rate is fed back in ack packets. The congestion window is updated for every incoming ack as per the original algorithms. The base RTT estimates are obtained in the RTT estimation routine.

We also modify the socket layer so that the file transfer application is able to select either TCP or RATCP as the underlying transport protocol. This enables us to compare the performance of competing TCP and RATCP sessions over the bottleneck link. Experiments involving random losses are carried out using the RLM in WALE. Files are transfered from "server" to "client". WALE is configured on the server. Along with the file transfer request, the client also requests the transport protocol (RATCP or TCP) to be used for the transfer. Throughputs are measured at the client.

V. Numerical Results

A. RATCP OldTahoe and TCP OldTahoe: Analysis and Simulation

The rate modulating Markov chain discussed in Section III.A is infinite. However, to arrive at the numerical results, we limit the number of ephemeral sessions to some finite number $M_{\max}$. We investigate the performance of RATCP and TCP with different *rates of variation of the available rate* to the tagged session. The arrival rate of ephemeral sessions along with $M_{\max}$

decides the average number of sessions on the link and variations in the available rate. With very high arrival rate, the number of sessions is almost always $M_{\max}+1$; hence each session gets a throughput of $C/(M_{\max}+1)$. Thus, in the ideal case the tagged session throughput varies between C and $C/(M_{\max}+1)$. The variance of the number of the ephemeral sessions increases with the arrival rate but decreases at higher arrival rates. Hence, the rate of variation of the rate available to the tagged session is low when the arrival rate is either very low or very high.

The common parameters selected for these results are: link rate, $C = 0.8$ Mb/s, link buffer, $B_{\max} = 10$ packets, TCP packet length $=$ 500 bytes, mean ephemeral session length, $\bar{L} = 200$ kB, round trip delay, $\Delta = 100$ ms, maximum number of ephemeral sessions on the link, $M_{\max} = 3$. Session arrival rate is given in sessions/s. For the results in this section the tagged session is assumed to know Δ and (delayed) rate feedback is made available to it every Δ.

Fig. 4 shows the basic comparison of the efficiency of RATCP and TCP obtained from the analysis. We define *efficiency* as the throughput of the tagged session normalized to the mean fair rate it gets. Let π_M denote the stationary probability distribution of the rate modulating Markov chain $\{M_k, k \geq 1\}$ discussed in Section III-A. Then $\sum_m (C/(m+1))\pi_M(m)$ is the mean fair rate of the tagged session. Hence,

$$\text{Efficiency} = \frac{\gamma}{\sum\limits_{m} \frac{C}{m+1}\pi_M(m)}.$$

Recall that γ denotes the throughput. An "ideal rate adaptive protocol" (IRAP) would adapt to the rate feedback instantaneously and without any losses. This way, efficiency can be interpreted as fraction of IRAP throughput obtained by a protocol under investigation.

There is an important effect of time scales of rate variations at the bottleneck link as compared to the round trip delay. When the rate variations are slow, feedback is effective and performance is expected to improve. On the other hand, the performance degrades because of rate mismatches; this effect is the worst when the bottleneck rate varies over propagation delay i.e., when the rate feedback is always "wrong". Recall that, since $M_{\max} = 3$ at any time t, when the arrival rate of the ephemeral sessions is very low or very high the fair rate variations are slow, whereas for intermediate arrival rates the rate variations are fast. We make the following observations from Fig. 4.

- When the arrival rate of the ephemeral sessions is very low, RATCP gives about 17%–20% better throughput than TCP. Since both RATCP and TCP recover conservatively from losses, the improvement with RATCP occurs since it suffers less losses, because of the adaptation to the rate.
- As the arrival rate increases RATCP does not have a significant advantage over TCP. This is because, when the rate variations are comparable to the propagation delay, there are frequent mismatches between the sending rate and the available bottleneck rate, and hence the rate feedback is not very effective. However, RATCP is able to contain packet losses to a smaller value as compared to TCP. Since the buffer backlog is only 1 packet in RATCP, it is not able to take advantage of the transient rate increases, and thus has a sharper decrease in the efficiency, with increasing ephemeral session arrival rate. The performance of RATCP can be enhanced in this region by a larger value of β. We show the performance with $\beta = 4$.
- When the arrival rate is higher, the mean number of sessions on the link increases. This implies that the rate available per session is small, and TCP needs to build a smaller window before a loss occurs. Thus the penalty for a packet loss is not significant and TCP performance is close to that of RATCP.
- When $\beta = 4$, RATCP is able to keep more packets in the network but without losses. Hence, with this value of β efficiency is improved substantially over the whole range of rate variation.

Fig. 5 shows the comparison of absolute throughputs (corresponding to efficiency results depicted in Fig. 4) of RATCP

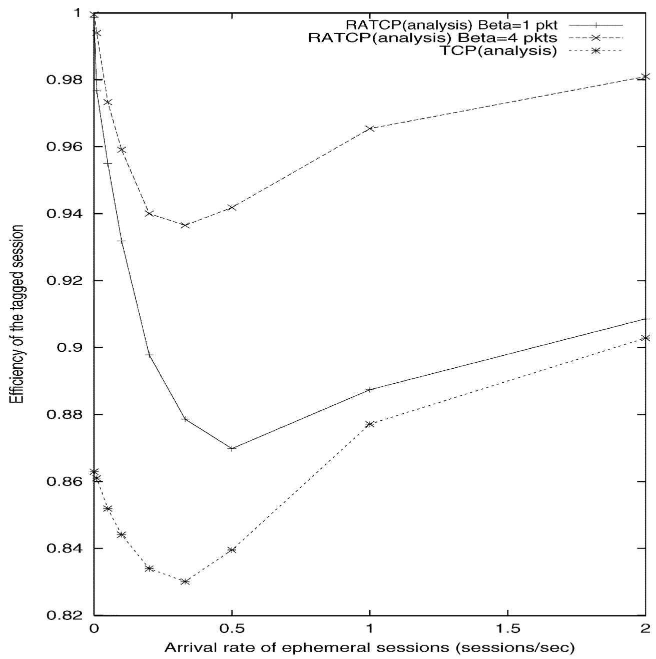


Fig. 4. Efficiency variation of RATCP and TCP with the ephemeral session arrival rate. Analysis.

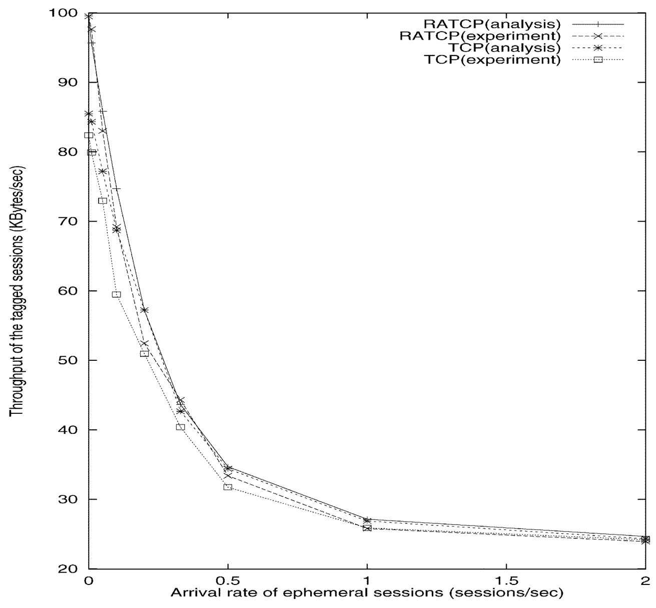


Fig. 5. Throughput variation of RATCP and TCP with the ephemeral session arrival rate. Analysis and experiment. $\beta = 1$ packet.

TABLE I
THROUGHPUT (kB/s) OF THE PERSISTENT SESSION FOR VARIOUS PROTOCOLS AND PARAMETERS. EACH COLUMN CORRESPONDS TO AN ARRIVAL RATE OF EPHEMERAL SESSIONS ON THE LINK

Ephemeral session arrival Rate / Protocol	0.0	0.05	0.1	0.2	0.5	1.0	2.0
RATCP (β=1):analysis	99.60	85.87	74.75	57.27	34.73	27.19	24.70
RATCP (β=1):experiment	99.52	83.05	69.24	52.50	33.43	25.83	23.97
TCP :analysis	86.34	77.23	68.75	57.27	34.44	26.93	24.37
TCP :experiment	82.44	73.02	59.52	51.00	31.79	25.93	24.24
RATCP (β=4):analysis	99.68	87.51	76.93	59.96	37.60	29.57	27.67
RATCP (β=4):experiment	99.52	83.62	71.58	57.73	35.63	29.07	27.26
RATCP-Reno (β=1):experiment	99.63	88.24	71.03	55.24	34.68	25.37	23.61
RATCP-Reno (β=4): experiment	99.54	87.13	72.97	58.96	36.58	29.67	27.90
TCP-Reno :experiment	92.31	77.14	65.24	49.63	34.19	27.61	25.87

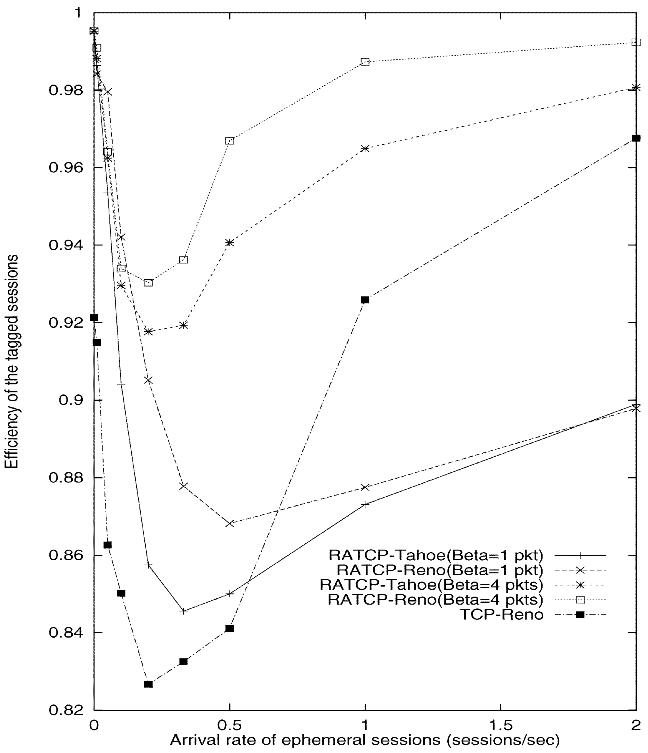


Fig. 6. Efficiency variation of RATCP (Reno and OldTahoe) with the ephemeral session arrival rate compared to TCP Reno.

and TCP obtained analytically as well as from experiments. Note from Fig. 5, that analytical and experimental results match well with analysis being slight overestimate. Overall the analysis procedure captures the performance quite well. Numerical values are shown in Table I.

B. RATCP Reno; Random Losses

We have described fast-retransmit and recovery in RATCP Reno in Section II. Table I gives the throughput comparison of RATCP Reno and TCP Reno. Fig. 6 shows the comparison of efficiency. Although TCP-Reno implements an efficient way of avoiding timeouts, it is, however, incapable of reducing losses. Hence, RATCP Reno (even RATCP OldTahoe, $\beta = 1$) outperforms TCP Reno when the rate variations are slow. However, fast retransmit works well in TCP Reno when the arrival rate of the ephemeral sessions is high; it matches the throughput of RATCP with $\beta = 1$ and is much more efficient (Fig. 6). Recall that in this region TCP needs to build a smaller window after a loss. This means that TCP Reno can keep more packets in the network as compared to RATCP ($\beta = 1$) and still recover from the losses efficiently. However, when the rate variations are fast, TCP loses multiple packets due to frequent rate mismatches. This leads to multiple window cutbacks.[3] Hence, TCP frequently recovers by timeout resulting in degradation of throughput. RATCP Reno, on the other hand, controls losses and implements much more efficient fast retransmit thereby performing overall better than TCP over a broad range of rate variations. Recall that, in RATCP *fast-retransmit and fast-recovery*, upon receiving three duplicate acks we set $W^{cong} \leftarrow \min(W^{cong}, W^{rate})$ instead of $W^{cong} \leftarrow (W^{cong}/2) + K$ as in TCP-Reno.

On links where transmission error probability is high, e.g., satellite links, it is particularly important that TCP retransmit the packets lost due to corruption without reducing its congestion window. Various techniques like FEC, ECN bits, ICMP messages, etc. have been proposed to inform TCP of the corruption losses [3]. However, it remains a difficult problem and major bottleneck in the performance of TCP over lossy links [23]. On the other hand RATCP-Reno maintains the fair window in fast retransmit; hence, it is indirectly able to differentiate congestion and corruption losses. This can be seen from Fig. 7 where we plot the throughput of a single persistent session versus the packet loss rate. The parameters are as given earlier except that there are no ephemeral session arrivals. Recall that the bottleneck link rate C is 100 kB/s and $\Delta = 100$ ms. $\beta = 1$ and 4. Notice that RATCP Reno succeeds in maintaining the throughput of the session above 85 kB/s for a wide range of packet loss probabilities, whereas the session throughput with TCP Reno drops to less than 50 kB/s with a packet loss probability of 1%. Further, with $\beta = 4$, RATCP does not achieve significantly higher throughput than when $\beta = 1$ due to only random loss. This is a significant result and suggests that RATCP could be used in conjunction with performance enhancing edge devices between the satellite networks and terrestrial networks.

C. Fairness

We continue to use the same experimental set-up and the link parameters. Fig. 8 shows the fairness comparison of RATCP and TCP, when two sessions with RTTs 100 ms and 200 ms share

[3]New TCP implementations do not cut the window multiple times if multiple packets are lost in the same round trip time.

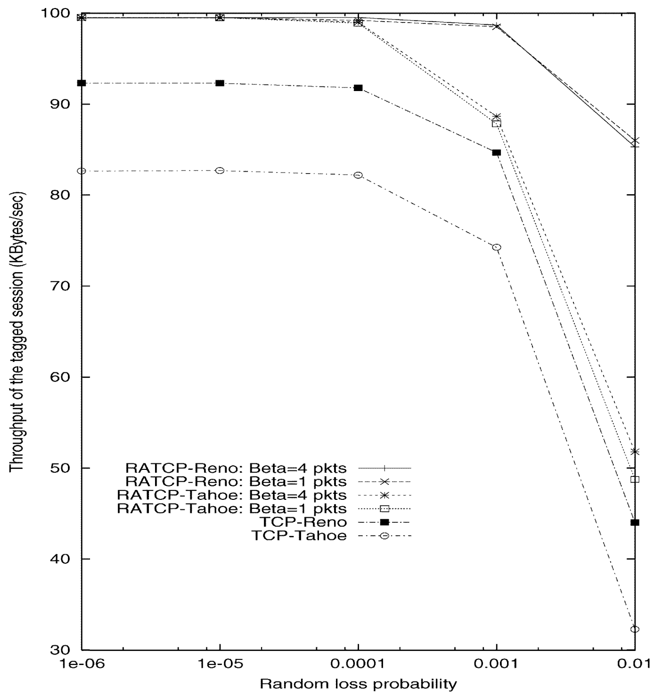


Fig. 7. Throughput variation of OldTahoe and Reno versions of RATCP and TCP with random packet drop probability.

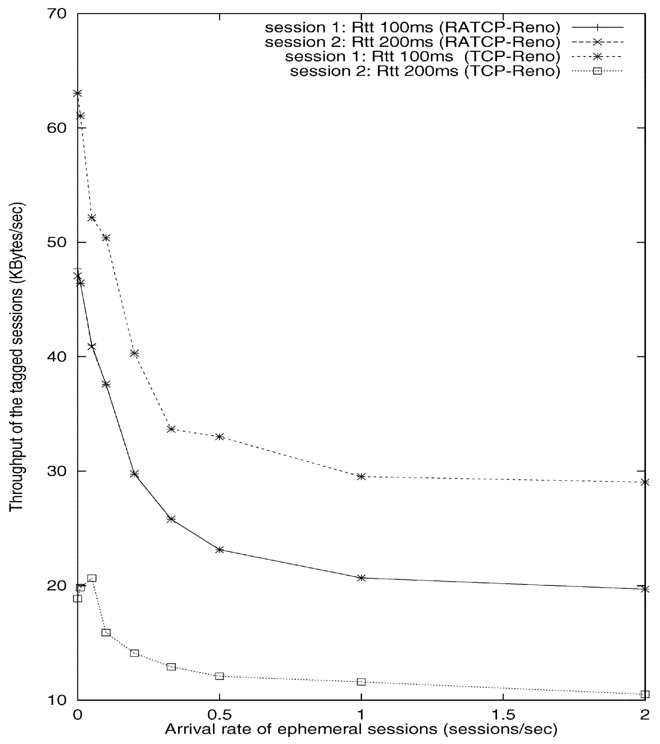


Fig. 8. Throughput comparison of 2 competing sessions on the link with different round trip times- 100 ms and 200 ms. Sessions either use RATCP Reno or TCP Reno. $\beta = 1$ packet.

the bottleneck link. When TCP sessions with larger propagation delays share a link with sessions with smaller delays they suffer because of the following effect—smaller delay sessions increase their windows at a higher rate creating frequent losses and recovery thereafter is slow for sessions with larger propagation delays. Since larger RTT session requires larger window

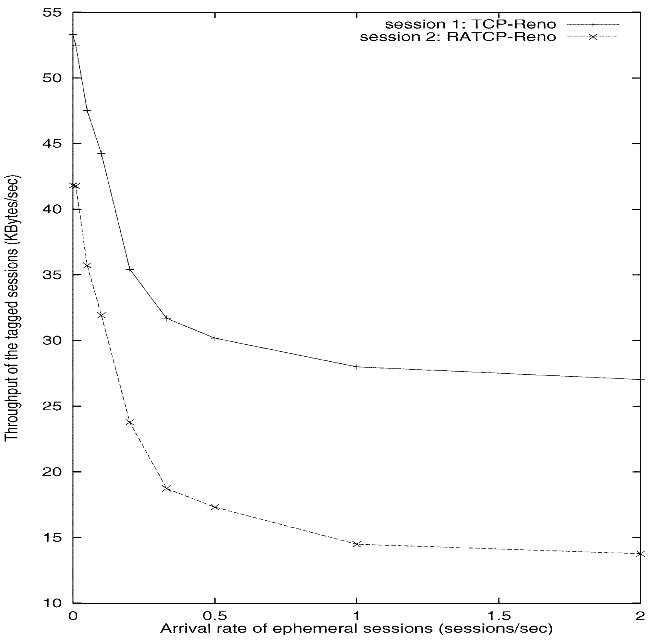


Fig. 9. Throughput comparison of two competing sessions on the link, one uses TCP Reno and the other RATCP Reno. RTT is equal to 100 ms for both the sessions. $\beta = 1$ packet.

for a given throughput the above phenomenon results in a low throughput for large RTT sessions. This leads to unfairness as seen from Fig. 8; session with 200 ms delay gets 50%–60% less throughput than the session with 100 ms delay. Since the windows are calculated based on the fair rate feedback, as expected, RATCP sessions in spite of different propagation delays get equal throughputs. Interestingly however, when an RATCP session competes with a TCP session, as seen from Fig. 9, TCP gains. This is because, RATCP limits its window, whereas TCP has more number of packets in the round trip pipe and creates losses for both. A similar phenomenon is seen when TCP-Vegas competes with TCP-Reno [24].

D. Finite-Size File Transfers (HTTP-Like TCP Transfers)

Web traffic is the predominant traffic in the Internet today. To model such a realistic situation, we need to consider a traffic model in which sessions arrive randomly and transfer small files. We assume that sessions arrive in a Poisson process and require file transfers with sizes exponentially distributed with mean 200 kB. If λ denotes the session arrival rate, then the load on the link, ρ, is defined as $\lambda\bar{L}/C$. Experiments are conducted for different values of ρ. The session arrival rate is then calculated from the formula for ρ. We now use the following parameters: link rate, $C = 2$ Mb/s, link buffer, $B_{\max} = 30$ kB, TCP packet length $= 1500$ bytes, mean file transfer size, $\bar{L} = 200$ kB, round trip propagation delay, $\Delta = 100$ ms. For RATCP, we assume that the exact rate is available at the sender (after one round trip time), and it uses the base RTT estimate to calculate the rate window.

Fig. 10 shows the variation of average throughput of sessions (averaged over 500 sessions) with the load on the link. Note that, the average throughput performance of RATCP and TCP is almost the same. However, it can be seen from Fig. 11 that losses incurred by RATCP with $\beta = 1$ are significantly fewer

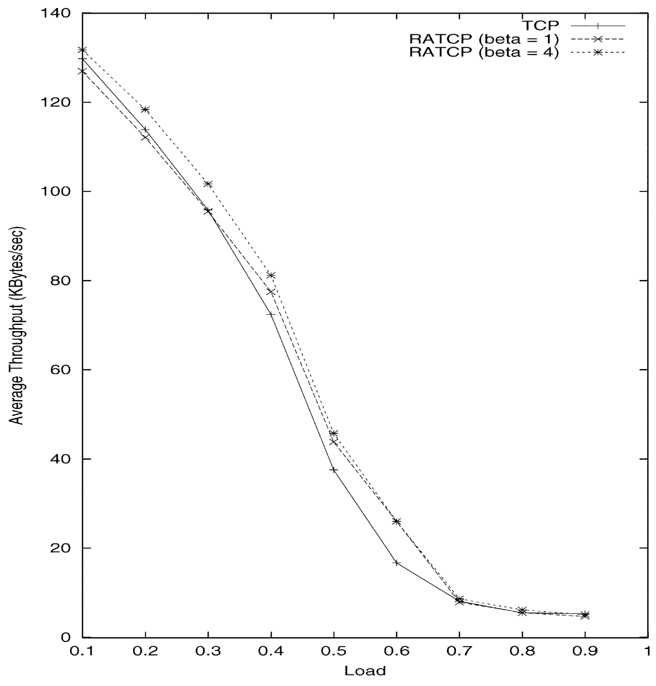


Fig. 10. Variation of average session throughput with load.

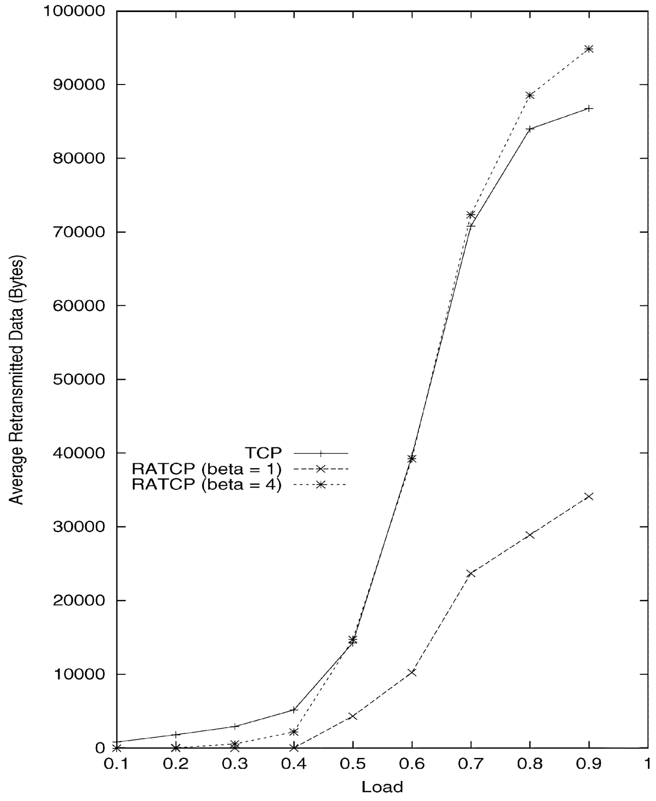


Fig. 11. Average retransmitted data per sessions versus load.

than by TCP and by RATCP with $\beta = 4$. This is explained by the following experimental observations. Fig. 12 shows the byte (transmitted on the link) numbers plotted against time for randomly picked 20 consecutive TCP sessions in the above experiment. The origin of time axis is the system time of the arrival of the first session and hence is arbitrary. The corresponding plot for RATCP sessions is shown in Fig. 13. Each curve represents the evolution of one session; the average throughput of

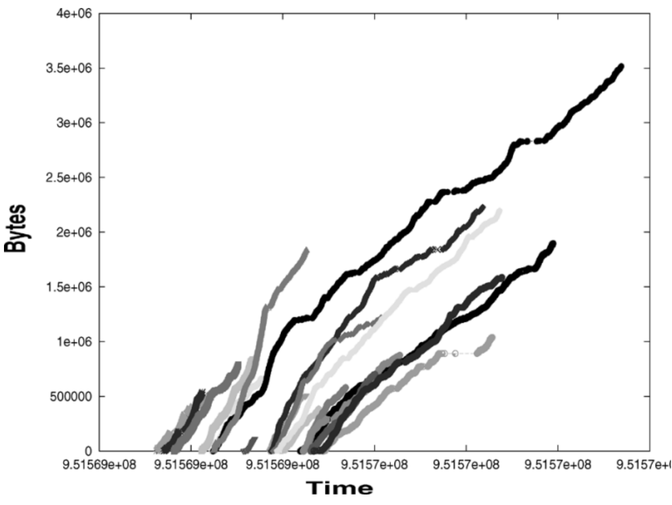


Fig. 12. Evolution of the congestion window of 20 TCP sessions sharing the link. Mean file transfer size is 200 kB.

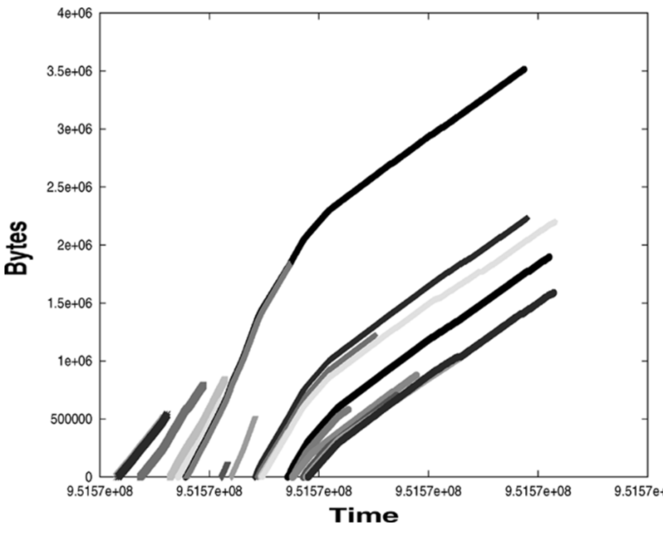


Fig. 13. Evolution of the congestion window of 20 RATCP sessions sharing the link. Mean file transfer size is 200 kB.

that session can be calculated as the ratio of total number of bytes transferred (obtained from y-axis) to the total time taken (obtained from x-axis). In addition, the slope of a curve gives the *instantaneous rate* that session gets. Note that, a discontinuity in a curve indicates packet losses. Observe that, the curves of RATCP sessions have equal slopes indicating fair allocation of instantaneous rates to all the sessions; hence RATCP sessions get equal instantaneous throughputs. As would be expected there are hardly any packet losses. On the other hand, TCP sessions incur more losses and some sessions get significantly higher throughputs than other sessions. We find that this leads to almost the same average performance of both the protocols.

Random Loss: Fig. 14 shows the performance of web-like transfers on a link with random losses. Load on the link is 0.5 and mean file transfer size is 200 kB. Two RTT's are studied 100 and 500 ms. Note that, RATCP gives a 10% improvement in the average throughput over TCP. With a large RTT, e.g., 500 ms typically encountered on satellite links, performance degrades significantly for both the protocols. However, we have found

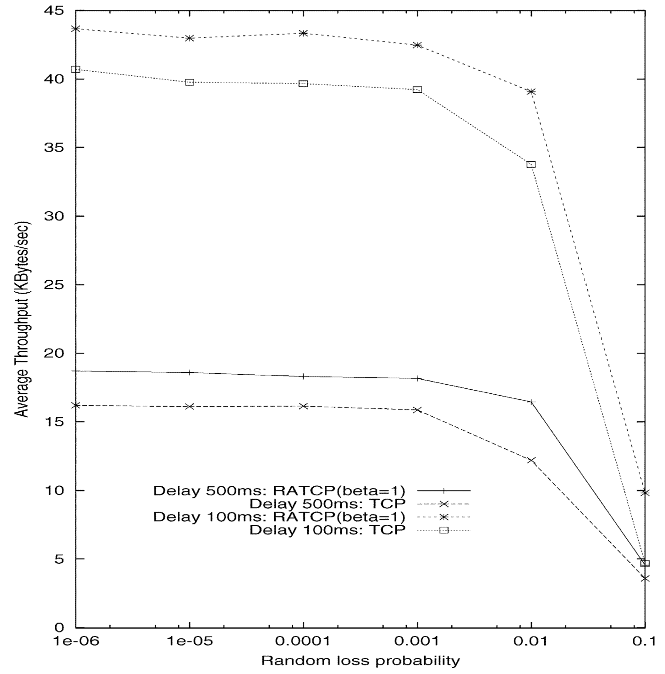


Fig. 14. Average sessions throughput versus random loss probability. Mean file transfer size is 200 kB. Round trip times are 100 ms and 500 ms.

that with larger file sizes (mean file size 1 MB) and small load values, RATCP maintains a higher throughout than TCP for a wide range of packet loss probabilities. This performance is similar to the one observed in Fig. 7.

VI. Available Rate Estimation and Feedback

To improve upon TCP's basic algorithms which estimate the available rate in the network, various algorithms have been proposed [25]. A sender-side technique uses the rate of returning acks. However, it assumes that the original spacing of packets is preserved in acks; this makes this technique problematic. This problem can be solved by estimating the available rate at the receiver using TCP packets and informing the source through a TCP options field. Using this information, the TCP source may set the value of slow start threshold or adapt the congestion window. However, this does not guarantee fairness since the estimated rate may not be the fair rate. In addition, the performance crucially depends on the accuracy of the estimates. Also, on lossy links these techniques will perform poorly.

The techniques discussed above use only the end-to-end information. A network based technique may be implemented using recently proposed Random Early Marking or (REM) [13]. REM is motivated by optimization based flow control, where sources and links maximize a global measure of network performance in a distributed fashion [19]. In a network of L links and S sources, a source s attains a utility $U_s(x_s)$ when it transmits at rate x_s. The objective is to maximize $\sum_s U_s(x_s)$ subject to the constraint that the total source rate on any link is less than the link capacity. To solve the optimization problem in a decentralised way, each link calculates a "price" per unit bandwidth by measuring the aggregate source rate. A source s is fed back the sum of the prices over all the links it uses; this sum called the path price for source s. It then chooses a transmission rate that maximizes its own utility based on the path price.

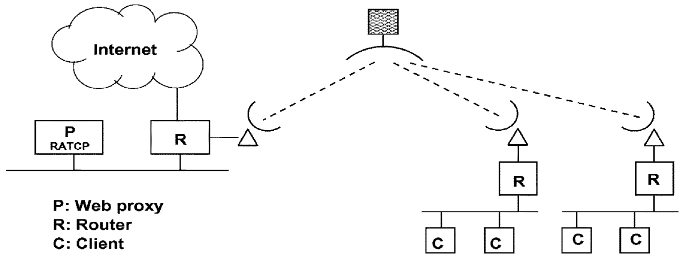


Fig. 15. A satellite networking situation where RATCP will be useful.

This is implemented as follows. Each link measures the total source rate using the buffer backlog and "feeds back" its price to the sources by "marking" packets. A packet is marked with probability that is *exponentially increasing in the price* so that the end-to-end marking probability is exponential in the path price. A source can estimate the price by the fraction of marked packets and then adjust its rate. Marking is done using ECN bits in IP packets. We have used $1/n$ as the fair share of bandwidth, however, with REM sources may use different utility functions to adjust their rates.

On wireless or satellite links, direct feedback from the edge devices can be obtained. With a split-connection approach ([26] and references therein), RATCP can be used on the wireless link. The edge device can explicitly calculate rates for sessions going through it and feed them to RATCP. A possible scenario is shown in Fig. 15 where clients download data from the Internet via a proxy server(shown as an integration of proxy-web server and a bandwidth controller) using a satellite link which is the bottleneck link. RATCP is implemented in the proxy for client side connections.

VII. Conclusions

In this paper we set out to understand the performance implications of feeding the available bottleneck link rate directly into TCP windows. Assuming that such information is available and there is a mechanism to feed it back to the source, we studied an approach for adapting the TCP congestion window to explicit rate feedbacks, and called this modification RATCP. Using analysis and an experimental test-bed we studied the performance of TCP and RATCP under various network scenarios. Our main observations are as follows.

1. There is an important effect of time scales of rate variations at the bottleneck link compared to the propagation delay. When the rate variations are slow compared to the RTT, feedback is effective and the performance is improved. On the other hand, when the propagation delay is large and rate variations are rapid, the performance degrades because of rate mismatches. These observations are similar to the ones in [14]. The context in [14] TCP over ATM-ABR; no feedback of rates to TCP windows.
2. When the file transfers are large and the load on the link is low, RATCP performs significantly better (17%–20%) than TCP.
3. RATCP is most advantageous in dealing effectively with random losses on the link. With the rate information, RATCP differentiates between congestion and corruption losses leading to higher throughputs over a wide range

of random loss probabilities. This scenario is particularly important from the point of view of wireless and satellite networks.

4. RATCP ensures fairness among sessions even if they have different propagation delays.
5. With short file transfers RATCP is only slightly better than TCP. TCP sessions incur more losses and some of the sessions get significantly higher throughputs than others. On the other hand, RATCP sessions get equal instantaneous throughput with hardly any packet losses.
6. It is possible to implement rate feedback mechanisms based on distributed flow control algorithms. With REM as the feedback mechanism ECN bit itself can be used.

VIII. Proof of Proposition 3.1

Let $P_X(x;x') = Pr(X_{k+1} = x'|X_k = x)$, where $x = (b, d, w^{cong}, w^{rate}, m)$ and $x' = (b', d', (w')^{cong}, (w')^{rate}, m')$. Define $w_+^{cong} = \min\{w^{cong}, w^{rate}\}$.

Remark: It requires too much notation to write down the complete transition probabilities for each pair of states. In the following exposition we adopt the approach of taking an event of interest, and obtaining the transition probability from one state to another if this event occurs. The sum of these probabilities yields the actual transition probability in the computer program that is used to obtain the stationary probability distribution of $\{X_k\}$.

We consider each of the three cases discussed in Section III-C.

Transition Probabilities for Case 1: During (t_k, t_{k+1}) the source is idle and there is no packet loss. Let $a = \lfloor \Delta \cdot C/(m+1) \rfloor$; we also have $W_{k+}^{cong} = w^{rate}$. Then it is clear that with probability $[P_M]_{mm'}$, $0 \le m' \le m+1$

$$X_{k+1} = (\max\{0, (b-a)\}, \min\{b, a\}, w^{rate}, a+\beta, m') \quad (4)$$

and

$$T_{k+1} = T_k + \Delta. \quad (5)$$

Model for Window Increase During Congestion Avoidance: We model the congestion avoidance phase probabilistically (see also [14], [27]). For a nonduplicate ack received at t, $W^{cong}(t)$ is incremented by 1 with probability $1/W^{cong}(t)$. To simplify the analysis, we assume here that the probability of window increase in (t_k, t_{k+1}) is constant and equals W_{k+}^{cong}.

Transition Probabilities for Cases 2 and 3: Recall the definitions of these cases from Section III-C. In Case 2 we have $d > h$, and only after h acks have been received will congestion avoidance resume; new packets will be generated and the window may increase. We call the acks that affect the window increase in (t_k, t_{k+1}), *effective acks*, and denote them by E_k (e is used to denote a particular value). With this notation (and recalling the notation for the elements of X_k) we have for Case 2, $e = d - h$, and for Case 3, $e = d$. Let N_k denote the total window increase in (t_k, t_{k+1}). Then for $0 \le \eta \le e$,

$$Pr(N_k = \eta | X_k = x) = \binom{E_k}{\eta} \left(\frac{1}{W_{k+}^{cong}}\right)^{\eta} \left(1 - \frac{1}{W_{k+}^{cong}}\right)^{E_k - \eta}. \quad (6)$$

Calculation of Loss Probability: A packet loss occurs when a packet finds the link buffer full (tail drop). It can be seen that loss occurs in (t_k, t_{k+1}) when the window increment is more than a value, called the loss threshold, n_{thresh}. We now obtain this threshold.

Let $\tau_1, \tau_2, \ldots, \tau_d$ denote the ack arrival epochs in (t_k, t_{k+1}).[4] With h as above, note that τ_{h+1} is the arrival epoch of the first effective ack. For $h = 0$, define $\tau_h = t_k$. Then for $j > h$, define B_k^j = buffer occupancy at τ_j, N_k^j = window increase in $(\tau_h, \tau_j]$, G_k^j = number of packets inserted into the link in $(\tau_h, \tau_j]$, and D_k^j = number of packets transmitted by the link in $(\tau_h, \tau_j]$. Note that, the number of packets inserted into the link till τ_j constitutes the packets triggered by acks received and the new packets generated due to the window increase up to and including epoch τ_j. For example in Case 3, $\tau_h = t_k$, and $G_k^j = j + N_k^j$ and, owing to the back-to-back assumption, $D_k^j = \min\{d, \lfloor j \cdot R_k/R_{k-1} \rfloor\}$.

Denote by b_{start}, the buffer occupancy at the first effective ack arrival, that is, just after the h^{th} ack arrives. For example, in Case 3, W^{cong} equals $b + d$, $h = 0$; then b_{start} equals b.

Consider the case that $b_{start} > 0$ and $R_{k-1} > R_k$ (note that, the loss may occur even when $R_k > R_{k-1}$ and $R_k \not> 2 \cdot R_{k-1}$). Then, for $j > h$, $B_k^j = \min(B_{\max}, (b_{start} + G_k^j - D_k^j))$. Let n_j be a particular value of N_k^j such that $(b_{start} + G_k^j - D_k^j) = B_{\max}$; hence, for $1 \le j \le e$, $n_j = \max(0, B\max - (b_{start} + j - \lfloor j \cdot R_k/R_{k-1} \rfloor))$. This means that if N_k^j exceeds n_j then loss occurs at τ_j. Observe that the sequence n_j is nonincreasing with j. Thus, in this case, we define

$$n_{thresh} = \max\left\{-1, B\max - \left(b_{start} + e - \left\lfloor \frac{e \cdot R_k}{R_{k-1}} \right\rfloor\right)\right\}. \quad (7)$$

It is easily seen that loss occurs if and only if $N_k > n_{thresh}$. Hence,

$$p_{loss}(x) = \sum_{i=n_{thresh}+1}^{e} \binom{e}{i} \left(\frac{1}{w_+^{cong}}\right)^{i} \left(1 - \frac{1}{w_+^{cong}}\right)^{(e-i)}. \quad (8)$$

Transition Probability Calculations: New Arrivals, No Loss: $\Rightarrow N_k \le n_{thresh}$. Recall that b_{start} is the buffer occupancy at the first effective ack arrival, and thus equals $\min(0, b - (h \cdot R_k/R_{k-1}))$, where, $h = b + d - w_+^{cong}$. Also $e = w_{k+}^{cong} - b$. From these, n_{thresh} is obtained by using (7). Let a_{eff} be the number of packets transmitted from the link in (T_k, T_{k+1}), and η the window increment in (t_k, t_{k+1}). Then

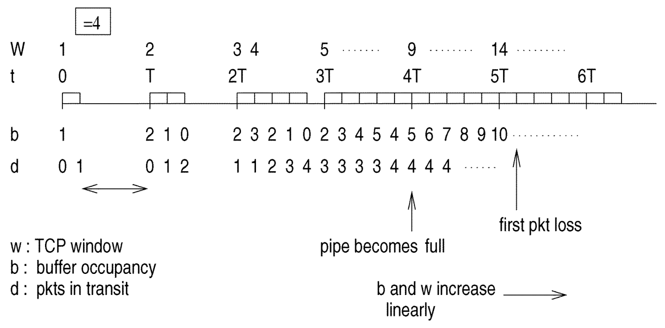
$$a_{eff} = \min\left\{a, \left(b - b_{start} + \min\left\{b_{start} + e + \eta, \left(\Delta - \frac{h}{R_{k-1}}\right)R_k\right\}\right)\right\},$$

where $a = \lfloor \Delta \cdot C/(m+1) \rfloor$ and it can be seen that, for $0 \le \eta \le n_{thresh}$ and $0 \le m' \le m+1$, with probability

$$\binom{e}{\eta} \left(\frac{1}{w_+^{cong}}\right)^{\eta} \left(1 - \frac{1}{w_+^{cong}}\right)^{(e-\eta)} [P_M]_{mm'}$$

$$X_{k+1} = (b', a_{eff}, b' + a_{eff}, a + \beta, m') \quad (9)$$

[4]The packets enter the link buffer *instantaneously* after an ack arrival; we, therefore, refer to a packet arrival epoch at the link by the corresponding ack arrival epoch at the source.



=4
W 1 2 3 4 5 ······· 9 ······· 14 ·········
t 0 T 2T 3T 4T 5T 6T
b 1 2 1 0 2 3 2 1 0 2 3 4 5 4 5 6 7 8 9 10 ···········
d 0 1 0 1 2 1 1 2 3 4 3 3 3 3 4 4 4 4 ······
first pkt loss
pipe becomes full
w : TCP window
b : buffer occupancy
d : pkts in transit
b and w increase linearly

Fig. 16. Slow start phase when the bandwidth-delay product is 4.

and

$$T_{k+1} = T_k + \Delta \tag{10}$$

where $b' := \min\left(0, b_{start} + e + \eta - (\Delta - (h/R_{k-1}))\, R_k\right)$.

Transition Probability Calculations: New Arrivals, Loss Occurs: $\Rightarrow N_k > n_{thresh}$. Recall that, we do not consider adaptation to the rate window in slow start.[5] Therefore, given *ssthresh* and the number of active sessions at the beginning of the recovery, the slow start duration and the state of the system at the end of slow-start can be determined.

Initial Conditions at Slow Start: Note that, $\{(W_k^{rate}, M_k), k \geq 1\}$ is a DTMC. Let, $Q(w, m; \bar{w}, \bar{m})$ denote the one-step probability. Then observe that

$$Q(w, m; \bar{w}, \bar{m}) = [P_M]_{m\bar{m}} 1_{\{\bar{w} = \lfloor \Delta C/(m+1) \rfloor + \beta\}}.$$

Recovery begins at t_{k+3}, *ssthresh* is set to W_{k+3}^{rate}, and during slow start the number of sessions constraining the persistent session's rate is assumed to be M_{k+3}; hence,

$$Pr(ssthresh = \bar{w}, M_{k+3} = \bar{m} | X_k = x) = Q^3(w^{rate}, m; \bar{w}, \bar{m}).$$

Modeling Slow Start: Note that, at the beginning of slow start, $b = 0$, $d = 0$, $w = 1$, $w^{rate} = \bar{w}$, $m = \bar{m}$. Since we assume that $ssthresh = \bar{w}$, and the rate for the persistent session is constant (given by $C/(\bar{m}+1)$) during slow-start, the slow start evolution is determined as follows. Define $\nu = \Delta \cdot C/(\bar{m}+1)$ and $F = \Delta + ((\bar{m}+1)/C)$. As in [28], we consider mini-cycles of duration F, where i^{th} mini-cycle refers to the time interval $[iF, (i+1)F)$. Fig. 16 shows the evolution of the buffer and the TCP window in the slow start phase. Let J denote the mini-cycle when the pipe becomes full. Then $J = \lceil \log_2(\nu + 1) \rceil$ and, $B((J+1)T) = 2^J - \nu + 1$ and $W^{cong}((J+1)T) = 2^J + 1$. Note that, if $B_{\max} < 2^{J-1} + 1$ then the packet loss takes place in or before the J^{th} mini-cycle (see [28]). If $B_{\max} > 2^{J-1} + 1$ then in every next mini-cycle, $W^{cong}(t)$ and $B(t)$ increase linearly by $\nu + 1$ leading to packet loss if $ssthresh - (2^J+1) + 2^J - \nu + 1 > B_{\max} \Rightarrow ssthresh - \nu > B_{\max}$. This causes a second period of timeout and recovery. *ssthresh* set for the second slow start does not exceed half the current value. Hence, the recovery is completed with the successful slow start phase. Results for the recovery phase can be arrived at with little algebra. The details are provided in [21].

[5]This basically means that, though the rate modulation process evolves independently during the period of recovery, we assume that the rate available to the tagged session remains constant during this period. This assumption is not made in the experiments.

These calculations provide the slow-start duration $L_{ss}(\bar{w}, \bar{m})$, and also the values of the number of packets in the buffer and in the delay queue at the end of slow start, i.e., $b_{ss}(\bar{w}, \bar{m})$ and $d_{ss}(\bar{w}, \bar{m})$. We will denote these simply by L_{ss}, b_{ss}, d_{ss}. Putting the above calculation together, we have, with probability (recall that we have $X_k = (b, d, w^{cong}, w^{rate}, m)$)

$p_{loss}(x) Q^3(w^{rate}, m; \bar{w}, \bar{m}) \, [P_M]_{\bar{m}, m'}^{L_{ss}}$

$$X_{k+1} = \left(b_{ss}, d_{ss}, b_{ss} + d_{ss}, \left\lfloor \frac{\Delta \cdot C}{\bar{m}+1} \right\rfloor + \beta, m'\right) \tag{11}$$

and

$$T_{k+1} = T_k + (3 + L_{ss})\Delta \tag{12}$$

with probability

$$p_{loss}(x) Q^3(w^{rate}, m; \bar{w}, \bar{m}).$$

It, therefore, follows from (4), (5), (9)–(12), that $\{X_k, k \geq 0\}$ is a Markov chain, since the distribution of X_{k+1} can be found without any knowledge of the past, given X_k. Also given X_k, the distribution of $T_{k+1} - T_k$ can be found without knowledge of the past. $\{(X_k, T_k), k \geq 0\}$ is thus a Markov Renewal Process. $\square$

REFERENCES

[1] L. Brakmo and L. Peterson, "TCP Vegas: end to end congestion avoidance on a global Internet," *IEEE J. Select. Areas Commun.*, vol. 13, no. 8, pp. 1465–1480, Oct. 1995.

[2] M. Mathis *et al.*, "TCP Selective Acknowledgment Options," Network Working Group, RFC 2018, 1996.

[3] S. Floyd. (2000) A Report on Some Recent Developments in TCP Congestion Control. [Online]. Available: http://www.aciri.org/

[4] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proc. ACM SIGCOMM*, Aug. 1996.

[5] S. Mascolo *et al.*, "TCP Westwood: Congestion Control with Faster Recovery," Univ. California, Los Angeles, Tech. Rep. CSD TR #200017, 2000.

[6] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Trans. Networking*, vol. 1, no. 4, pp. 397–413, Aug 1993.

[7] T. Bonald *et al.*, "Analytic evaluation of RED performance," in *Proc. IEEE INFOCOM*, Mar. 2000.

[8] M. Christiansen *et al.*, "Tuning RED for web traffic," in *Proc. ACM SIGCOMM*, Aug. 2000.

[9] V. Firoiu and M. Borden, "A study of active queue management for congestion control," in *Proc. IEEE INFOCOM*, Mar. 2000.

[10] S. Floyd, "TCP and explicit congestion notification," *Comput. Commun. Rev.*, vol. 24, no. 5, pp. 10–23, Oct. 1994.

[11] K. Pentikousis and H. Badr, "An evaluation of TCP with explicit congestion notification," *Ann. Telecommun.*, 2003, to be published.

[12] T. Ott. (1999, May) ECN protocols and TCP paradigm. [Online]. Available: http://www.icir.org/floyd/ecn.html

[13] S. Athuraliya *et al.*, "Random early marking," *Proc. QofIS*, Sep. 2000.

[14] S. Shakkottai *et al.*, "TCP performance over end-to-end rate control and stochastic available capacity," *IEEE/ACM Trans. Networking*, vol. 9, no. 4, pp. 377–391, Aug. 2001.

[15] P. Narvaez and K.-Y. Siu, "An acknowledgment bucket scheme for regulating TCP flow over ATM," in *Proc. IEEE* , Nov. 1998.

[16] L. Kalampoukas *et al.*, "Explicit window adaptation: A method to enhance TCP performance," in *Proc. IEEE INFOCOM*, Mar. 1998.

[17] R. Satyavolu *et al.*, "Explicit Rate Control of TCP Applications," ATM Forum, Doc. No. ATM-Forum/98–0152R1, 1998.

[18] S. Karandikar *et al.*, "TCP rate control," *Comput. Commun. Rev.*, vol. 30, no. 1, Jan. 2000.

[19] S. Low and D. Lapsley, "Optimization flow control-I: Basic algorithm and convergence," *IEEE/ACM Trans. Networking*, vol. 7, no. 6, pp. 861–874, Dec. 1999.

[20] S. Abraham and A. Kumar, "A new approach for asynchronous distributed rate control of elastic sessions in integrated packet networks," *IEEE/ACM Trans. Networking*, vol. 9, no. 1, pp. 15–30, Jan. 2001.

[21] A. Karnik, "Performance of TCP Congestion Control with Rate Feedback: TCP/ABR and Rate Adaptive TCP/IP," M.Eng. thesis, Indian Institute of Science, Bangalore, 1999.

[22] A. Anvekar. (2000) WALE: A Wide Area Link Emulator on a Linux PC. ERNET Project, IISC Tech. Rep.. [Online]. Available: http://ece.iisc.ernet.in/netlab/

[23] M. Allman *et al.*, "Ongoing TCP Research Related to Satellites," Network Working Group, RFC 2760, 2000.

[24] J. Mo *et al.*, "Analysis and comparison of TCP reno and vegas," in *Proc. IEEE INFOCOM*, Mar. 1999.

[25] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *Proc. ACM SIGCOMM*, Aug. 1999.

[26] A. Ewerlid, "Reliable communication over wireless links," in *Nordic Radio Symp. (NRS)*, Sweden, Apr. 2001.

[27] A. Kumar, "Comparative performance analysis of versions of TCP in local network with a lossy link," *IEEE/ACM Trans. Networking*, vol. 6, no. 4, pp. 485–498, Aug. 1998.

[28] T. V. Lakshman and U. Madhow, "The performance of TCP/IP for networks with high bandwidth delay products and random loss," *IEEE/ACM Trans. Networking*, vol. 5, no. 3, pp. 336–350, Jun. 1997.

Aditya Karnik received the the Ph.D. degree from the Indian Institute of Science, Bangalore, in 2004. He is currently a Postdoctoral Fellow at the University of Waterloo, Waterloo, Canada.

His research interests are performance evaluation, optimization and control of communication networks.

Mr. Karnik was a recipient of the IBM Research Fellowship.

Anurag Kumar received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, Kanpur, and the Ph.D. degree from Cornell University, Ithaca, NY.

He was with Bell Laboratories, Holmdel, NJ, for over six years. Since 1988, he has been with the Department of Electrical Communication Engineering, Indian Institute of Science (IISc), Bangalore, where he is now a Professor, and is also the Chairman of the department. He is a coauthor of the textbook *Communication Networking: An Analytical Approach* (Morgan Kaufmann, 2004). His area of research is communication networking, specifically, modeling, analysis, control and optimization problems arising in communication networks and distributed systems.

Dr. Kumar is a Fellow of the Indian National Academy of Engineering (INAE) since 1998. He serves on the editorial board of IEEE Communications Surveys and Tutorials.