Performance of TCP Congestion Control with Explicit Rate Feedback: Rate Adaptive TCP (RATCP)

Aditya Karnik and Anurag Kumar
Dept. of Electrical Communication Engg.
Indian Institute of Science, Bangalore, 560012, INDIA
email: karnik, anurag@ece.iisc.ernet.in

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 the performance of RATCP (using analytical models, and an experimental test-bed) in various network scenarios, and to compare the performance of RATCP and TCP, both with and without fast-retransmit and fast-recovery. We find that when sessions with the same round trip times share a bottleneck link then, even with ideal fair rate feedback, the performance of RATCP is only slightly better than that of TCP. RATCP, however, does reduce losses significantly. When there are random losses, however, RATCP with fast-recovery provides substantially better throughput than plain TCP. Further, when sessions with different round trip times share the bottleneck link, as expected, RATCP ensures fairness. Finally, we suggest a practical situation in which RATCP can be useful for improving web access performance.**

Keywords—**explicit rate control; rate adaptive TCP; bandwidth sharing; adaptive window congestion control**

I. Introduction

In the Internet, TCP (Transmission Control Protocol) uses an adaptive window mechanism for sender-receiver flow control, for congestion control, and for enforcing bandwidth sharing. The window adaptation is based on *implicit feedbacks* from the network; acknowledgements cause the congestion window to increase, and packet losses (indicated by timeouts or duplicate acknowledgements) cause the window to decrease. Owing to this blind rate adaptation mechanism, TCP has often been found to be inefficient and unfair in its throughput performance. Recent research has, therefore, focussed on explicit involvement of the network in the congestion control mechanism of TCP. This work includes router based mechanisms such as packet drop policies and explicit congestion notification ([1], [2]), ack pacing by the network edge device ([3]), explicit window feedback based on buffer occupancy ([4]) or rate allocation at the network edge device ([5]).

In this paper, we consider explicit feedback of fair session rates from the network *directly to the individual TCP sources*, and study a policy for utilizing this rate information in TCP's adaptive window based congestion control. We call this modification Rate Adaptive TCP (RATCP). It can be expected that, if the TCP sources adapt to the rate feedback there will be fewer losses, the network bandwidth will be used efficiently, and fairness will be achieved among the competing sessions. We assume that the network is somehow able to feedback fair session rates to TCP sources (later in the paper we suggest a practical situation in which this can be done). The TCP transmitters adapt their congestion windows based on this rate feedback and a round-trip-time (RTT) estimate. Thus our concern in this paper is to study the performance implications of feeding 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.

We compare the performance of RATCP and TCP in the following scenarios: (1) 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; thus the persistent session has a time varying fair rate. (2) A persistent session over a bottleneck link with random loss. (3) Two persistent sessions with different round-trip times sharing a bottleneck link; *both* the sessions use RATCP or TCP. (4) Two persistent sessions on a link, one using RATCP and the other TCP. (5) A link being shared by ephemeral sessions that randomly arrive and depart.

We use an analytical model and an experimental test-bed to study the above scenarios. For case (1) we develop an analytical model for obtaining the throughput of the persistent session using either RATCP or TCP, both without the fast-retransmit feature. The analysis is based on identifying a certain Markov renewal reward process, and calculating the TCP throughput as the reward rate in this process. Our experimental setup comprises an implementation of RATCP in Linux; the bottleneck link is emulated in the Linux kernel. This setup is used to verify the analysis, and to provide numerical results for the other cases, including results for RATCP and TCP with fast-retransmit and recovery.

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

Based on research supported by Nortel Networks.
Address all correspondence to Anurag Kumar.

session is time varying ($R(t)$), and $\hat{\Delta}(t)$ is an estimate (at the transmitter) 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 transmitter at time t. In this paper, we wish to study how an appropriate implementation of 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 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 $W^{cong} \leftarrow \frac{W^{cong}}{2} + K$ as in TCP-Reno), and the missing packet is retransmitted. After every additional acknowledgement received W^{cong} is increased by 1. Upon receipt of the ack for the resent packet, congestion avoidance resumes, as described above.

We call this modification of TCP, *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 it is approximate, is very useful for providing insight into factors that affect the performance of a protocol, and, in addition, analysis can help to validate and debug simulations and experimental implementations.

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. When there are m ephemeral sessions, we assume that these sessions use exactly $\frac{m}{m+1}$ of the link capacity of C packets per sec, and the persistent session's share is $\frac{C}{m+1}$ pkts/sec. Thus the fair bandwidth available to the persistent session is randomly time varying.

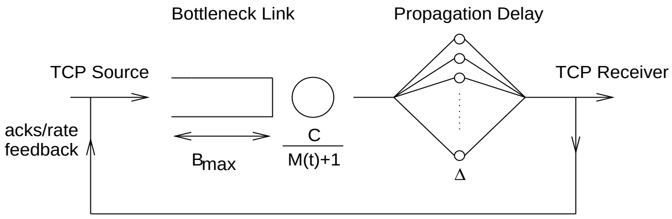

Fig. 1. A queueing model of the persistent TCP session.

Figure 1 shows a schematic queueing model of the persistent TCP session. Note that the ephemeral sessions are only modelled 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.

The continuous time processes are hard to analyse. Instead, we follow the analysis procedure developed in [6]. 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 Figure 2), and define

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

where, at epoch t_k, W_k^{rate}, 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.

We model $\{M_k, k \geq 0\}$ as a Markov chain that evolves independently of the other components of the process $\{Z_k\}$. We assume that delay Δ is *known* at the TCP source. Owing to one Δ delay in rate feedback, the rate window calculated at t_k is given by $W_k^{rate} = \lfloor R_{k-1}.\Delta \rfloor + \beta$ where, $R_{k-1} = \frac{C}{M_{k-1}+1}$. Then, the window adaptation policy implies that $W_{k+}^{cong} = \min\{W_k^{cong}, W_k^{rate}\}$ (note that $k+$ denotes "just after epoch t_k").

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

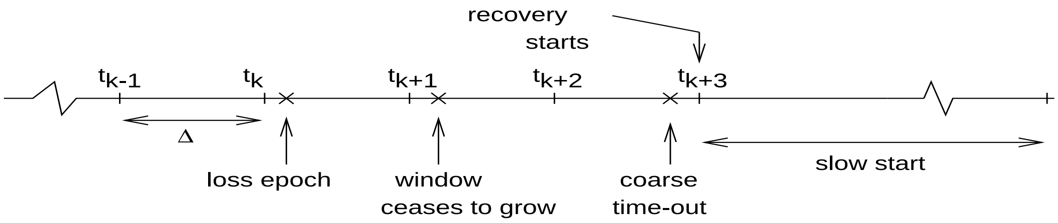

Fig. 2. Evolution of the process $\{Z_k,\ k \geq 0\}$, showing the model for timeout based loss recovery..

- The transmitter 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 acknowledgements (acks) arrive at the TCP transmitter 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 Figure 2). We assume that this ack arrives at the source in (t_{k+1}, t_{k+2}). At this point the transmitter 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 Figure 2). Recalling that we are not modelling 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.

A Markov renewal process: Define, the embedded epochs $\{T_k\}$ by: $T_0 = t_0$, and for $k \geq 0$

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

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 $X_0 = Z_0$. With the assumptions made above, it can be argued that $\{X_k, k \geq 0\}$ is a Markov chain. The analysis proceeds by obtaining the epochs T_k and the Markov chain $X_k = Z_{T_k}$. The transition probabilities of this Markov chain are obtained by examining each of the three cases described above. We model the linear increase during the congestion avoidance probabilistically. For simplicity in the analysis, we do not consider adaptation to W^{rate} in the slow start phase; however, W^{rate} is used to set the value of $ssthresh$ at time-out. We model the losses in slow start. It can then be shown that, given X_k and T_k, the distribution of T_{k+1} can be computed without any knowledge of the past. Hence, the process $\{(X_k, T_k), k \geq 0\}$ is a Markov renewal process (MRP). It is this MRP that is our model for the persistent TCP connection with time varying bandwidth.

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 (see [**?**]) we have,

$$\gamma = \frac{E_\pi V}{E_\pi U} \tag{1}$$

where, $p_{loss}(x)$ denoting the probability of packet loss in the state $X_k = x$,

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

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

The details of this analysis and analysis of TCP without rate feedback are provided in [7].

IV. Simulation Setup

The simulation results for the network of Figure 1 reported here are obtained from a Linux based test-bed, where the bottleneck link (buffering, transmission rate, and propagation delay) is emulated in the loop-back interface. Rate variations on the bottleneck link are artificially created by the rate transitions made to occur at discrete time epochs. Loop-back file transfers are run within one machine, using the actual TCP code modified according to RATCP. These modifications include the introduction of a variable that determines the phase of the TCP algorithm. Note that, we do consider the adaptation to the rate window in slow start (unlike the assumption made in the analysis). In order to validate the analysis, the exact rate feedback is artificially provided to the TCP sender. In case of multiple connections, we modify the ftp client and the socket layer so that the client application is able to select the underlying transport protocol (TCP or RATCP). This enables us to compare the performance of competing TCP and RATCP sessions over the bottleneck link.

V. Numerical Results

A. RATCP OldTahoe and TCP OldTahoe: Analysis and Simulation

The common parameters selected for these results are: link rate, C= 0.8 Mbps, link buffer, B_{max}, = 10 packets, TCP packet length = 500 Bytes, mean ephemeral session length, $\bar{L}$, = 200 KBytes, round trip delay, Δ, = 100 ms, maximum number of ephemeral sessions on the link, M_{max}, = 3. In order to validate the analysis, it is assumed that the round trip time is known to the sender.

Figure **??** shows the basic comparison of the performance of RATCP and TCP obtained analytically as well as from simulations. Note from Figure **??**, that analysis and simulation results match well with analysis being slightly overestimating. Thus, overall the analysis procedure captures the performance quite well. Numerical values are shown in Table I.

Note 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. 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. 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.

We provide additional results in Table I. These numbers show the effect of increasing β, and reducing the link buffer size. It can be seen that the performance of RATCP is improved in the region where the fair rate varies rapidly by a larger value of β since RATCP is able to take advantage of transient rate increases. If the buffer size is reduced from 10 to 8 packets, observe that, because of the control over the number of packets in the network, the reduction in the buffers does not degrade the performance of RATCP as much as TCP.

B. RATCP Reno; Random Losses: Simulation Results

In the following simulations, RATCP uses the base RTT *estimate* to calculate the rate window.

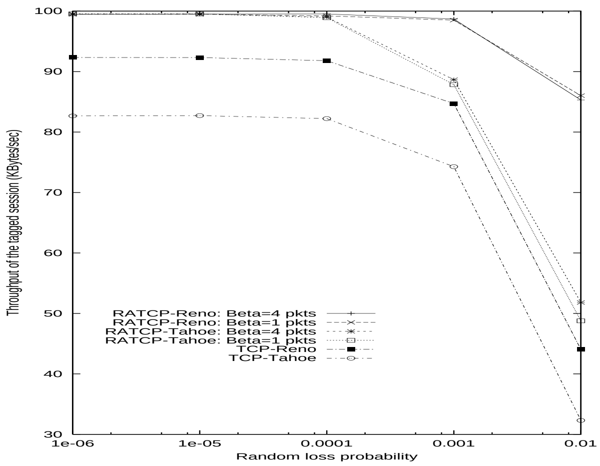

Fig. 3. Throughput variation of Tahoe and Reno versions of RATCP and TCP with random packet drop probability.

We have described fast-retransmit and recovery in RATCP Reno in Section II. Table I shows the comparison of RATCP Reno and TCP Reno. Fast retransmit works well in TCP Reno when the rate variations are slow; particularly with high arrival rate it matches the performance of RATCP with $\beta = 1$. However, multiple losses in the fast variation region cause multiple window cutbacks and hence the degradation in throughput. RATCP Reno on the other hand improves overall performance.

On links where transmission error probability is high, e.g. satellite links ([8]), it is particularly important that TCP retransmit the packets lost due to corruption without reducing its congestion window. Since RATCP-Reno maintains the fair window in fast retransmit, it is indirectly able to differentiate congestion and corruption losses which is a difficult problem for TCP. This can be seen from Figure 3 where we plot the throughput of a single persistent session versus the packet loss rate. The parameters are as in Table **??**, except that there are no ephemeral session arrivals. Notice that RATCP Reno succeeds in maintaining the throughput of the session above 85KBytes/sec (on a 100KBytes/sec link) for a wide range of packet loss probabilities, whereas the session throughput with TCP Reno drops to less than 50KBytes/sec with a packet loss probability of 1%.

Ephemeral session arrival Rate(sessions/sec) / Protocol	0.0	0.01	0.05	0.1	0.2	0.33	0.5	1.0	2.0
RATCP (β=1):analysis	99.60	95.69	85.87	74.75	57.27	43.68	34.73	27.19	24.70
RATCP (β=1):simulation	99.52	97.68	83.05	69.24	52.50	44.35	33.43	25.83	23.97
TCP :analysis	85.54	84.36	77.23	68.75	57.27	42.71	34.44	26.93	24.37
TCP :simulation	82.44	79.95	73.02	59.52	51.00	40.43	31.79	25.93	24.24
RATCP (β=4):analysis	99.68	97.33	87.51	76.93	59.96	46.55	37.60	29.57	27.67
RATCP (β=4):simulation	99.52	96.83	83.62	71.58	57.73	46.69	35.63	29.07	27.26
RATCP (B_{max}=8 pkts):analysis	99.60	93.46	83.62	74.59	57.08	43.29	34.43	26.95	24.65
RATCP (B_{max}=8 pkts):simulation	99.54	93.41	78.90	71.17	48.86	45.87	32.98	25.39	23.86
TCP (B_{max}=8 pkts):analysis	83.13	81.49	74.92	66.43	55.30	41.06	33.16	26.11	23.57
TCP (B_{max}=8 pkts):simulation	78.91	77.40	73.51	65.35	47.19	38.24	31.08	25.23	23.47
RATCP-Reno (β=1):simulation	99.63	95.89	88.24	71.03	55.24	45.00	34.68	25.37	23.61
RATCP-Reno (β=4): simulation	99.54	98.19	87.13	72.97	58.96	47.12	36.58	29.67	27.90
TCP-Reno :simulation	92.31	87.67	77.14	65.24	49.63	43.42	34.19	27.61	25.87

TABLE I

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

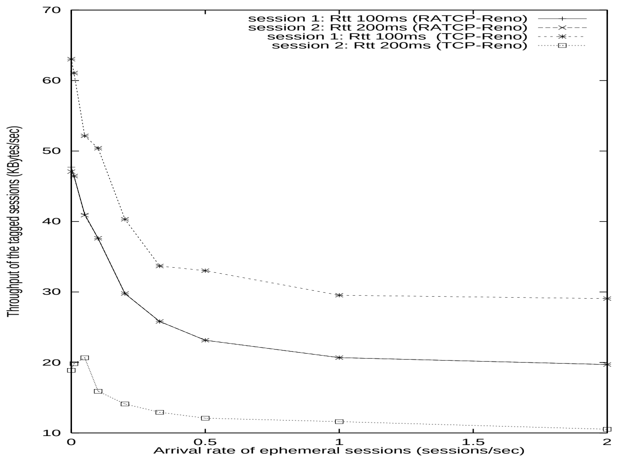

Fig. 4. Throughput comparison of 2 competing sessions on the link with different round trip times- 100ms and 200ms.Sessions either use RATCP Reno or TCP Reno. $\beta = 1$ packet.

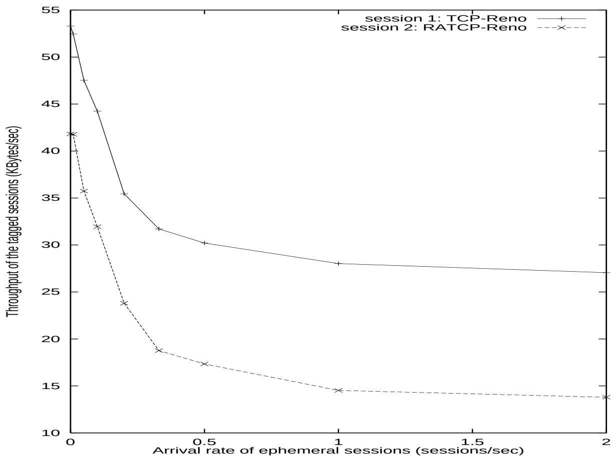

Fig. 5. Throughput comparison of 2 competing sessions on the link, one uses TCP Reno and the other RATCP Reno. Round trip time is equal to 100ms for both the sessions. $\beta = 1$ packet.

C. Fairness: Simulation Results

We continue to use the same simulation set-up and the link parameters. Figure 4 shows the fairness comparison of RATCP and TCP, when 2 sessions with round trip times 100ms and 200ms share the bottleneck link. As expected, RATCP is seen to be fair as against TCP which is biased against sessions with larger propagation delays. However, when an RATCP session competes with a TCP session, as seen from Figure 5, TCP gains because of its greedy nature. A similar phenomenon is seen when TCP-Vegas competes with TCP-Reno (see [9]).

D. Small file transfers: Simulation Results

Web traffic is the predominant traffic in the Internet today. To model such a realistic situation, we need to consider the transfer of small files. We model file sizes as being exponentially distributed with mean 200KB. We now use the following parameters: link rate, C, = 2 Mbps, link buffer, B_{max}, = 30KBytes, TCP packet length = 1500 Bytes, mean file transfer size, $\bar{L}$, = 200 KBytes, round trip Delay, Δ, = 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.

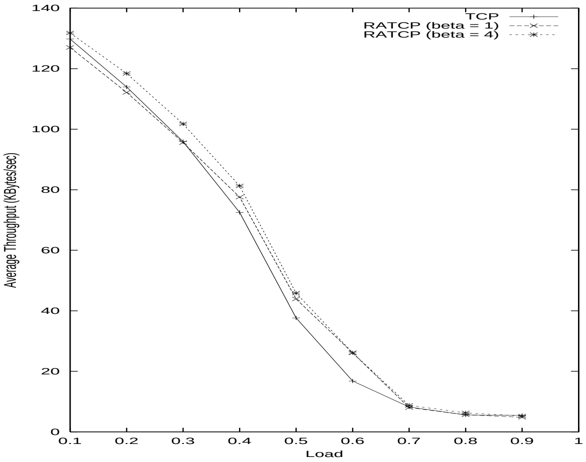

Fig. 6. Average throughput variation of sessions vs load. Mean file transfer size is 200KB.

Figure 6 shows the variation of average throughput of sessions (averaged over 500 sessions) as the load on the link increases. Note that, the performance of RATCP and TCP is almost the same. Since RATCP allocates fair rates to all the sessions, sessions get equitable throughputs, but with TCP smaller files get significantly higher throughputs and the average performance of both the protocols is almost the same. Figure 7 shows that losses incurred by RATCP with $\beta = 1$ are significantly fewer than by TCP and by RATCP with $\beta = 4$.

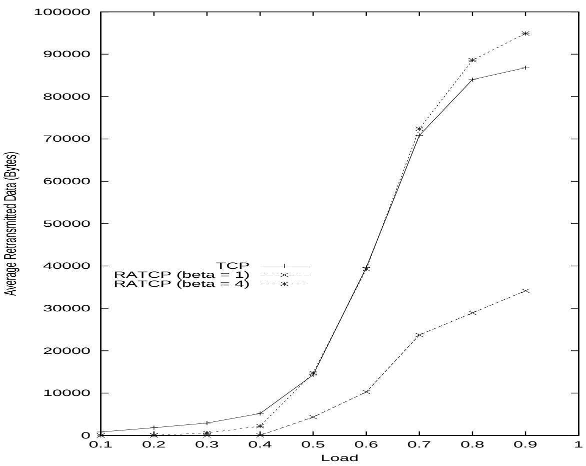

Fig. 7. Average retransmitted data per sessions vs load. Mean file transfer size is 200KB.

Experiments with small file transfers on a link with random losses show that RATCP gives 10% better throughput than TCP. However, with larger file sizes (mean file size 1 MByte) and small load values, performance similar to Figure 3 can be achieved.

VI. Conclusions

We studied an approach for adapting the TCP congestion window to explicit bottleneck rate feedbacks, and called this modification RATCP. Analysis and simulation results show that though the throughput performance RATCP is only slightly better than TCP, it reduces losses substantially and also deals effectively with random losses on the link. Further, RATCP ensures fair bandwidth sharing between sessions even if they have different propagation delays.

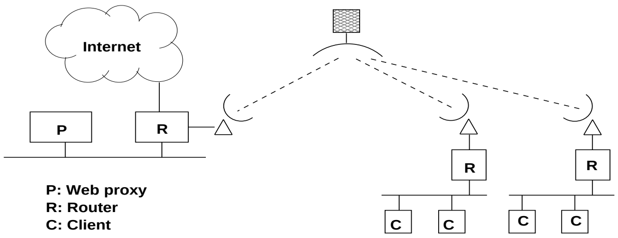

Fig. 8. A satellite networking situation where RATCP will be useful.

We believe that RATCP can find application in bandwidth managers. A possible scenario is shown in Figure 8 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. Ongoing simulations of this network scenario show that RATCP performs significantly better than TCP under low load conditions when the download file sizes are larger.

References

[1] Sally Floyd and Van Jacobson, "Random early detection gateways for congestion avoidance.," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, August 1993.

[2] K.K. Ramakrishnan and Sally Floyd, "A proposal to add explicit congestion notification ECN to IP.," Tech. Rep., IETF Draft, September 1998.

[3] Paolo Narvaez and Kai-Yeung Siu, "An acknowledgement bucket scheme for regulating TCP flow over ATM," in *Proc. IEEE Globecom 1998*, 1998.

[4] L. Kalampoukas, Anujan Varma, and K.K. Ramakrishnan, "Explicit window adaptation: A method to enhance TCP performance," in *IEEE Infocom 1999*. IEEE, March 1998.

[5] R. Satyavolu, Ketan Duvedi, and S. Kalyanraman, "Explicit rate control of TCP applications," Tech. Rep. ATM Forum Document Number: ATM-Forum/98-0152R1, February 1998.

[6] S.G. Sanjay and Anurag Kumar, "TCP over end-to-end ABR: A study of TCP performance with end-to-end rate control and stochastic available capacity," in *Proc. IEEE Globecom 1998*, 1998.

[7] Aditya Karnik, "Performance of TCP Congestion Control with Rate Feedback: TCP/ABR and TCP/IP," M.S. thesis, Indian Institute of Science, January 1999.

[8] Mark Allman et. al, "Ongoing TCP research related to satellites," Tech. Rep. IETF Draft: draft-ietf-tcpsat-res-issues-05.txt, May 1999.

[9] Jeonghoon Mo, Richard J. la, Venkat Anantharam, and Jean Walrand, "Analysis and comparison of TCP Reno and Vegas," in *IEEE Infocom 1999*. IEEE, March 1999.

[10] Ajit Anvekar, "WALE: A wide area link emulator on a linux PC," Tech. Rep. ERNET Project, IISc, May 1999.