TCP Rate Control [1]

Shrikrishna Karandikar, Shivkumar Kalyanaraman, Prasad Bagal[2], Bob Packer [3]
Department of ECSE, Department of Computer Science,
Rensselaer Polytechnic Institute.
karans@cs.rpi.edu, pbagal@us.oracle.com,
shivkuma@ecse.rpi.edu, bob@packeteer.com
Ph: 518-276-8979

Abstract

This paper presents TCP Rate Control, a new technique for transparently augmenting end-to-end TCP performance by controlling the sending rate of a TCP source. The sending rate of a TCP source is determined by its window size, the round trip time and the rate of acknowledgments. TCP rate control affects these aspects by modifying the ack[4] number and receiver window fields in acknowledgments and by modulating the acknowledgment rate. From a performance viewpoint a key benefit of TCP rate control is to avoid adverse performance effects due to packet losses such as reduced goodput and unfairness or large spread in per-user goodputs. Further, TCP rate control positively affects performance even if the bottleneck is non-local and the end-host TCP implementations are non-conforming. These aspects are demonstrated through a comparative study of TCP rate control, RED and TCP-ECN. The TCP rate control approach has been implemented and patented by Packeteer Inc.

1 Introduction

TCP congestion control is designed for network stability, robustness and opportunistic use of network resources on an end to end basis [9]. Using a robust technique to detect packet loss (timeout or triple-duplicate acks [21]), TCP infers congestion and trades off per-user goodput for network stability. Specifically, TCP throughput is known to be a function which is inversely proportional to the round trip time, the timeout delays and the square root of loss probability [17] (ignoring effects of small windows and timeouts [15]):

$$T_i \propto \frac{1}{\sqrt{p_i}.RTT_i} \tag{1}$$

where,
T_i = throughput for flow i.
p_i = probability of a packet loss for flow i.
RTT_i = round trip time for flow i.

Given this equation, we can view the function of any buffer management algorithm managing TCP flows as assigning loss probabilities (p_i) and queuing delays (which affect RTT_i) to competing TCP flows in order to meet performance requirements such as utilization, queuing delays, spread of per-user goodputs etc. However, this equation assumes that the TCP receiver window is not a limiting factor, which is not necessarily the case. Therefore if the TCP receiver window were the primary limiting factor, we could design a buffer management algorithm in which TCP throughput would not depend primarily on loss rate p_i (or RTT_i) under controlled operating conditions.

The design of TCP rate control is motivated by the above observations. It uses rich congestion-related information available at the bottleneck and calculates rate allocations for competing TCP flows. It then enforces these allocations by modifying the ack number and receiver window fields in the ack headers, and by modulating the ack rate. Observe that the rate allocation can be done by leveraging any of the well-known rate-calculation algorithms studied for the ATM

[1] This work has been sponsored by Packeteer, Inc, and in part by NSF Contract number ANI9806660 and DARPA contract number F30602-97-C-0274
[2] Prasad Bagal is with Oracle, Inc., CA, USA.
[3] Bob Packer is with Packeteer, Inc., CA, USA.
[4] Abbreviations: ack for acknowledgment, RTT for round trip time, dupack for duplicate ack, SACK for selective acknowledgment.

ABR service [12]. Such algorithms have already been shown to achieve strong control of metrics such as utilization, queuing delay and fairness of allocations. The novel part in TCP rate control is to ensure that these properties are preserved in the rate enforcement process.

The key contributions and observations of this paper are:

- Section 1.1 articulates the issues in the rate-to-window translation and enforcement process . Sections 3 and 4 discuss related work and deployment issues respectively. Specifically, TCP rate control is best deployed at WAN edges of enterprise or ISP networks and not in ISP network cores.
- We also study the performance of TCP rate control in comparison to RED and ECN [7, 19] (section 2).
  - We divide our performance measures into two groups: *user metrics and provider metrics*. We find that in general, RED and ECN optimize provider metrics while TCP rate control optimizes both user and provider metrics.
  - For long file-transfers, TCP rate control can manage the spread of per-user goodputs (a user metric) without compromising on utilization, aggregate goodput and queuing delays (provider metrics) even if the round trip times (RTTs) of the competing flows are different (section 2.2). We have verified the same result for other parameter dimensions. For brevity, we have included these results in a more detailed technical report [25].
  - For short-file transfers both TCP rate control and TCP-ECN manage the spread of transfer times (a user metric) better compared to RED. But TCP-rate control does so by slightly reducing the total number of transfers completed (a provider metric), whereas TCP-ECN trades off a larger queue (a provider metric) for the same benefit (section 2.3).
- Another key benefit of TCP rate control is in managing misbehaving (non-conforming) TCP sources which can be easily created by hacking operating systems like Linux. Such sources if unchecked can combine to steal bandwidth from conformant sources (*denial-of-service attacks*), or even lead to congestion collapse. Assuming that such sources respond to ack header fields (receiver window and ack number), TCP rate control can successfully restrain them and eliminate all their ill-effects (section 2.4). RED and ECN are much less effective in this respect.
- TCP rate control can be reasonably effective even if the bottleneck is non-local, i.e. it is implemented not at the bottleneck but appears elsewhere in the path of flows. However, in this case it loses some of the advantages of controlling packet losses (section 2.5).

1.1 TCP Rate Control

The two core ideas behind TCP rate control are:

1. Calculate a rate allocation per-TCP flow, and
2. Enforce the rate-allocation by manipulating TCP header fields and the ack rate.

1.1.1 Rate Allocation Algorithm

For commercial reasons we do not disclose the Packeteer rate allocation algorithm. Instead, we have used a well-known *max-min fair* algorithm due to Anna Charny [4](see Algorithm 1) in our simulations. The claims made in this paper therefore apply to the general TCP rate-control approach because any other max-min fair rate allocation algorithm can be used in the place of the algorithm used here. For example, in an earlier study, we used a different rate-allocation algorithm ERICA [23, 24].

Algorithm 1 Rate Allocation Algorithm

The goal of this algorithm is to allocate max-min fair rates to competing TCP flows. The algorithm described here is a minor variant of that proposed by Charny [4]. Initially, equal rate allocations are given to all competing flows. Then sending rates of flows are estimated by maintaining an exponential average over a *rate sampling interval*. When a flow does not to utilize its allocation, it is labeled as *bottlenecked*. Excess allocation is stripped off from all such *bottlenecked* flows and allocated to non-bottlenecked or *hungry* flows. This step is repeated until there is no residual bandwidth to allocate or all flows are bottlenecked. This algorithm is invoked every time a new flow begins, a flow terminates or when the rate sampling interval expires. The resulting rate allocations are stored into a table. A stepwise description of the algorithm follows.

1. For each flow i, let R_i be the measured rate and A_i be the allocated rate.
2. If N is the total number of flows and B is the bottleneck capacity, the initial allocation for each flow i is
$$A_i = \frac{B}{N}. \tag{2}$$
3. If $R_i < p.A_i$ for some satisfaction percentage p (a statically chosen parameter), then mark flow i as *bottlenecked*, else mark it as *hungry*.
4. Let U be the aggregate residual bandwidth i.e the bandwidth which remains unutilized by the bottlenecked flows.
5. Distribute this residual bandwidth evenly over all the hungry flows. If H is the total number of hungry flows , the new allocation for a hungry flow j is given by,
$$A_j = A_j + \frac{U}{H} \tag{3}$$
6. For each bottlenecked flow k the new allocation is given by,
$$A_k = \frac{A_k + R_k}{2} \tag{4}$$
So for bottlenecked flows, the allocation approaches the measured rate over successive iterations of the algorithm.

The choice of the *rate sampling interval* mentioned above involves a tradeoff between reliability of measurements (longer intervals) and speed of response (shorter intervals). Ideally, the interval length should be at least the maximum of the following: the longest RTT and the time required to see at least one packet from each active flow [24]. In the case of extremely short lived flows (that last for lesser duration than the sampling interval) some prior knowledge of the sending rate may be used. Even though we tested with short lived flows in our simulation (section 2.3), they lasted for at least a few RTTs and hence easily outlived the rate sampling interval chosen according to the previously mentioned guidelines.

1.1.2 Rate-enforcement algorithm

Once a rate r has been calculated, it can be converted into a window value as follows:

W = r × T

An obvious choice for T is the round trip time of the flow, not including queuing delays, RTT_i (measurement of RTT_i is discussed later). An earlier unpublished study of ours also shows that several other choices (eg:a fixed T

Algorithm 2 Rate Enforcement Algorithm

This algorithm enforces the rate allocation by converting the rate to a window value. Additionally, it spaces out the acknowledgments of a flow, so that they are evenly distributed over its RTT.

1. For each flow i let ,
   w_i = the calculated window size in units of packets. Changes at the end of every rate sampling interval
   $\triangle_i$ = the inter-ack spacing in seconds. Changes at the end of every rate sampling interval
   RTT_i = the round trip time (RTT) of flow i, ideally not including queuing delays, in seconds.
   MSS_i = the maximum segment size of flow i, in bytes.
   A_i = the rate allocation in bytes/s. Changes at the end of every rate sampling interval.

2. Observe that for each flow i, we have:
   $$w_i \times MSS_i = A_i \times RTT_i \tag{5}$$
   So the window value can be calculated as,
   $$w_i = \frac{A_i \times RTT_i}{MSS_i} \tag{6}$$

3. The inter-ack spacing time (RTT_i/w_i) can also be obtained from equation 5:
   $$\triangle_i = \frac{RTT_i}{w_i} = \frac{MSS_i}{A_i} \tag{7}$$

4. For each flow i, acks (if available) are clocked out at intervals of $\triangle_i$ with the receiver window field set to W_i. Burstiness is introduced when W_i fluctuates or when it is is not an integral multiple of MSS. This burstiness is not smoothed out in our modeling, but is smoothed out in the Packeteer implementation.

5. The receiver window field of the ack of flow i is set as:
   $$W_i = min(w_i \times MSS_i \text{ , actual receiver window)} \tag{8}$$

6. The ack number field in the header is determined based upon two variables (max and min sequence number) used to denote the ack queue. In the common case, the ack number would be chosen to progress by one MSS_i. Adjustments for dupacks, piggy-backed acks and SACKS are minor.

---

estimate for all flows) lead to unfairness [23]. This window value W is then used to limit the TCP congestion window variable. The feedback is given via the receiver window field (Wr) in the TCP header:

Wr_{new} = Min (Wr_{old}, W)

Note that if the window scale option is used, the router must be aware of the window scale factor, and should adjust the 16-bit window feedback appropriately. Further, the TCP checksum should be adjusted as follows:

Delta = $Wr_{new} - Wr_{old}$ (one's complement subtraction)

TCP_checksum = TCP_checksum + Delta (one's complement addition)

The header modification does not violate the TCP protocol. Hence the scheme is transparent and requires no standardization. However, the need for reading and writing into TCP headers means that the IP payload should not be encrypted or authenticated. Further, the need to measure the round trip time (RTT_i) of each flow means that the solution is applicable to network edges. The RTT of a flow can be approximated by observing packets and corresponding acks, or vice versa. No extra traffic is injected. Ignoring these details, Algorithm 2 (above) presents the rate enforcement part of TCP rate control.

Observe that TCP rate control paces the acks over a round trip time (steps 3 and 4 above) in addition to the receiver window modification. This is because, given a fixed window size, the rate of a TCP source is dictated by the rate of acknowledgments received. Therefore control of the ack rate in general smoothes out burstiness in TCP transmission.

Another source of burstiness occurs when there is a change in the rate allocation which manifests itself as a change in window size, or when the ideal window size w_i for rate regulation is not an integral multiple of MSS_i (step 4). Such window changes at the end of rate sampling intervals could potentially result in a burst of packets (if the window increased) or in an idle period (if the window was reduced). By distributing the window changes over multiple ack transmission opportunities, even such burstiness can be smoothed out (not modeled in our simulations). Similarly, rounding up of non-integral window values might result in errors especially when average window sizes are small (eg: low speed bottlenecks, large number of flows). However, for simplicity, we do not model these aspects, though Packeteer's implementation addresses them. This leads to some excess queues in simulation, especially in cases such as low-speed bottlenecks with large number of flows. This problem has also been articulated by Robert Morris in his study of effects caused by large number of TCP flows [15].

Another interesting aspect of TCP rate control is that, since the TCP window is the sum of the packets in the forward direction and the acks in the reverse direction, larger ack queues can be traded off for smaller packet queues. Moreover, the entire per-flow ack queue can be maintained in by holding just three variables: two of which indicate an interval of sequence numbers being stored (`left_edge` and `right_edge`), and one indicating the flow's MSS (maximum segment size). The `left_edge` is the highest ack number that has been forwarded to the sender and `right_edge` is the highest ack number the receiver has sent so far. This is equivalent to queuing (`right_edge` - `left_edge`) acks. This makes the space requirement for the ack queue constant , since physical queuing of acks is not necessary. Acknowledgments are then generated at a suitable rate using the recorded information (step 4).

In the case of piggy-backed acks the ack information in the data packet is first recorded and then the ack number and receiver window fields in the packet header are modified suitably before forwarding it. For duplicate acks an additional variable is used to count the number of dupacks. Then while sending the last ack in our ack queue (containing sequence number `right_edge`) we generate as many dupacks as indicated by this counter (since all dupacks received so far would be numbered `right_edge`). Since SACKS are a special case of duplicate acks, they are handled similarly with addition of a few more variables to record the block number information. Our simulations did not contain piggy-backed acks (unidirectional flows) or SACKS (end systems implemented TCP-Reno).

2 Performance Analysis

In this section, we compare the performance of TCP Rate Control with RED and TCP-ECN [7, 19]. A brief discussion of parameters, metrics and the base configuration used follows.

2.1 Metrics, Parameters and Configurations

We classify our metrics into two major categories: *user/customer metrics* and *operator/provider metrics* as described below:

Provider metrics: The common provider metrics we use in all simulations are *average link utilization, average queue length, maximum queue length, and total number of packet drops.* In addition for short file transfer simulations, we introduce a new provider metric: *the number of transfers completed*.

User metrics for large file transfers: We would like to measure the per-flow goodput, which is the rate at which the destination application receives information. It excludes the retransmission rate, but includes the effect of delays such as retransmission and timeout delays. Instead of measuring N such metrics, we summarize it using two metrics: *the average (per-flow) goodput and the coefficient of variation (ratio of the standard deviation and the average)* which measures the relative spread in per-user goodputs.

User metrics for short file transfers: Since average per-user goodput is meaningless for short-transfers, we use transfer time instead. As before, we use the average and the coefficient of variation of this quantity.

We evaluate performance with local and non-local bottlenecks. Figure 1 gives the basic configuration template that we used in our simulations used in the case of local bottlenecks with long file-transfers (section 2.2), short file-transfers (section 2.3) and misbehaving sources (section 2.4). It contains a single bottleneck shared by a set of unidirectional TCP flows. This simple template matches typical corporate network WAN connections where the expensive WAN link or leased line is the key bottleneck in the system. For the heterogeneous RTT simulations, the sources were grouped into four groups, each group having a different RTT (specifically 300us, 3ms, 30ms and 300 ms). In case of RED the minimum and maximum queue thresholds were 10 and 30 respectively, the queue averaging parameter w_q was set to 0.02 and the drop probability p was 0.2.

The end systems implemented *TCP Reno*. This is because, the overwhelming majority of hosts today use TCP Reno, even though newer versions of TCP (New-Reno and SACK) are being deployed. We would expect the performance gap to reduce with these newer versions, but have not evaluated them in this paper.

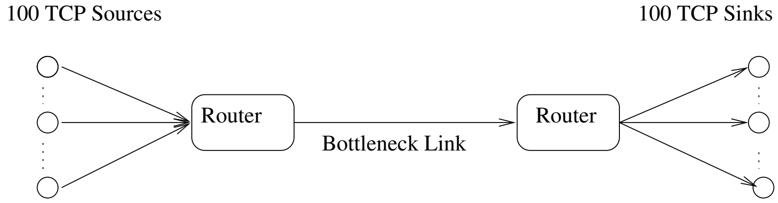

Figure 1: Configuration template used in simulations with local bottlenecks

We have evaluated the following parameters dimensions in our simulations for the local-bottleneck case: speed of the bottleneck (56 kbps - 45 Mbps), the RTTs (300 us - 300 ms), schemes used (TCP rate control, RED, ECN), and number of flows (10 - 100). The workloads used include short and long file transfers and misbehaving (non-conformant) TCP sources.

2.2 Long File Transfers, Heterogeneous RTTs

In this section we demonstrate that the TCP rate control when operating at the bottleneck can decouple TCP throughput from loss rate (and to some extent, RTT), without compromising provider metrics. To illustrate this point, we have chosen the RTTs of flows in the configuration to be heterogeneous.

2.2.1 RED (heterogeneous RTTs)

Table 1 lists the performance metrics with RED in a heterogeneous RTT configuration. Observe the last column which is bold faced. The coefficient of variation (CoV) indicates a large spread (or variation) of per-user goodputs. In other words, a randomly chosen flow has a non-trivial probability of getting worse performance (in terms of goodput) compared to the mean. This occurs because RED allocates different loss probabilities (p_i), and the flows themselves have different RTTs. But we have also observed similar behavior with homogeneous RTTs [25], indicating that the component of CoV attributable to loss probabilities is non-trivial.

In terms of the other metrics, the utilization is high (94-99%), queuing delays are low (10-30 packets), but the number of packet losses (552-4418 packets) and its effect on aggregate goodput increase with the speed of the bottleneck (eg: in the last row, the aggregate goodput on a 45 Mbps line is 32 Mbps). In other words, RED optimizes performance in terms of provider metrics, trading off user metrics under these conditions.

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**Drop** Pkts	**μ Goodpt** Mbps	**Coeff. of variation (CoV)**
0.056	98.79	30.33/62	552	0.00049	**1.18367**
0.384	99.68	27.54/62	890	0.00313	**0.79872**
1.5	99.85	21.80/63	1453	0.01252	**1.26278**
10	99.54	14.56/63	2626	0.08803	**1.99602**
45	94.17	9.59/61	4418	0.32083	**1.51962**

Table 1: RED, 100 sources, Heterogeneous RTTs. Result: Optimized provider metrics. Tradeoff: High CoV, low avg. goodput.

2.2.2 ECN (heterogeneous RTTs)

Table 2 shows the performance of ECN under similar conditions (observe last column, bold-faced). ECN reduces the coefficient of variation (CoV) considerably, and eliminates the effect of packet losses on user metrics, while trading off slightly higher queuing delays (226-313 packets).

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**Drop** Pkts	**μ Goodpt** Mbps	**Coeff. of variation (CoV)**
0.056	97.90	313.21/564	0	0.00052	**0.55769**
0.384	99.66	414.75/640	0	0.00242	**0.28512**
1.5	99.91	252.28/300	0	0.01497	**0.03273**
10	99.99	252.26/299	0	0.09995	**0.07494**
45	100.00	226.45/286	0	0.45004	**0.23940**

Table 2: ECN, 100 sources, Heterogeneous RTTs. Result: Reduced CoV, higher goodput than RED. Tradeoff: High queuing delays

2.2.3 TCP rate control (heterogeneous RTTs)

Table 3 shows the corresponding performance of TCP rate control. Observe that TCP rate control further diminishes the coefficient of variation (CoV) and the queuing delays. The comparative performance in terms of CoV is illustrated in Figure 2. Though we have emphasized CoV in this evaluation, the important point to note is that TCP rate control optimizes *all* metrics of interest i.e. *both* user and provider metrics. As described earlier, this is possible due to the direct control of the receiver window and the ack rate.

We have verified the same result for all other parameter dimensions such as homogeneous RTTs, LAN vs WAN RTTs, and for different numbers of competing flows. For brevity, we have excluded these results, but they may be found in a more detailed technical report online [25].

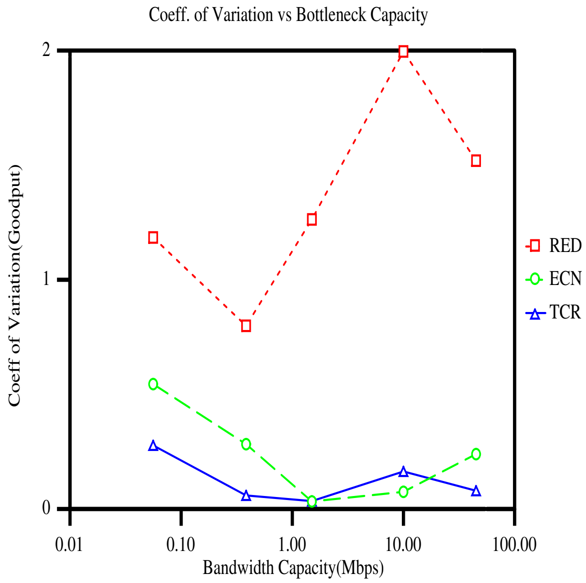

Figure 2: Plot for coefficient of variation of per-flow goodput - Heterogeneous RTTs, long transfers

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**Drop** Pkts	**μ Goodpt** Mbps	**Coeff. of variation (CoV)**
0.056	97.60	256.67/420	0	0.00046	**0.28261**
0.384	98.64	105.19/194	0	0.00372	**0.05914**
1.5	99.91	99.34/177	0	0.01496	**0.03476**
10	99.99	89.63/177	0	0.09995	**0.16378**
45	100.00	91.87/154	0	0.45000	**0.07962**

Table 3: TCP rate control, 100 sources, heterogeneous RTTs. Result: Optimized User AND provider metrics. Trade-off: None.

2.3 Short File Transfers, Homogeneous RTTs

In this section we look at the performance of the schemes with respect to short file transfers. The same configuration template (Figure 1) is used in these simulations. Flows are grouped into four sets and the start time of every set is staggered by 250 ms. Every flow sends 10K bytes (10 packets) and closes the connection. After a pause of 250 ms, the same source reopens the connection (with parameters set to initial values) and sends another 10K bytes (10 packets) and so on. The simulation time was 100 seconds.

As mentioned earlier, since goodput is meaningless for short-transfers, we use per-user transfer time (average and coefficient of variation) and the total number of transfers as our metrics. A similar model has been used in the past [1]. Though this is not a model of WWW transfers, it may provide illustrative information because WWW transfers are typically short.

Our primary observation in this section is that for short-file transfers, TCP rate control and TCP-ECN manage the spread of transfer time (a user metric) better when compared to RED. But TCP rate control does so by slightly reducing the total number of transfers completed (a provider metric), whereas TCP-ECN trades off a larger queue (a provider metric) for the same benefit. Our results for TCP rate-control were affected in this case by some modeling approximations.

2.3.1 RED (Short transfers)

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**# Trans**	μ T. time Milli Secs	**Coeff. of variation (CoV)**
0.056	99.87	30.45/53	52	75925	**0.40706**
0.384	99.88	23.37/50	436	16797	**1.23062**
1.5	99.16	12.68/49	1702	4848	**3.57821**
10	97.09	7.33/51	6655	234	**1.04665**
45	89.32	5.47/47	7195	161	**0.48968**

Table 4: RED, 100 sources, Short Transfers. Result: Higher number of transfers. Tradeoff: Higher CoV

Table 4 tabulates the results for simulations with RED. We observe that the average transfer time reduces with increase in bandwidth. The coefficient of variation (CoV) of per-user transfer times is large indicating that some transfers were able complete at the expense of other transfers. In other words, even though a larger number of transfers are completed, a randomly chosen transfer would have a non-trivial probability of having a larger transfer time when compared to the mean.

2.3.2 ECN (Short transfers)

Table 5 shows the simulation results of short transfers with ECN.

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**# Trans**	μ T. time Milli Secs	**Coeff. of variation (CoV)**
0.056	99.77	687.48/960	25	99624	**0.00282**
0.384	99.86	218.86/616	355	22294	**1.03563**
1.5	99.95	220.27/273	1816	5090	**0.14240**
10	99.96	89.74/153	7256	573	**0.05464**
45	87.65	4.97/77	7272	152	**0.04307**

Table 5: ECN, 100 sources, Short Transfers. Result: Lower CoV. Tradeoff: Queuing delay

We note that the average response times in slower speed configurations are slightly worse than RED because, in spite of not having loss-triggered effects, the queuing delays add significantly to the response time. Except for queuing delays at low speeds, ECN performance scaled well with increase in link speeds allowing a larger number of transfers and reduction in coefficient of variation of transfer time.

2.3.3 TCP rate control (Short transfers)

The results for TCP rate control in this situation are shown in table 6.

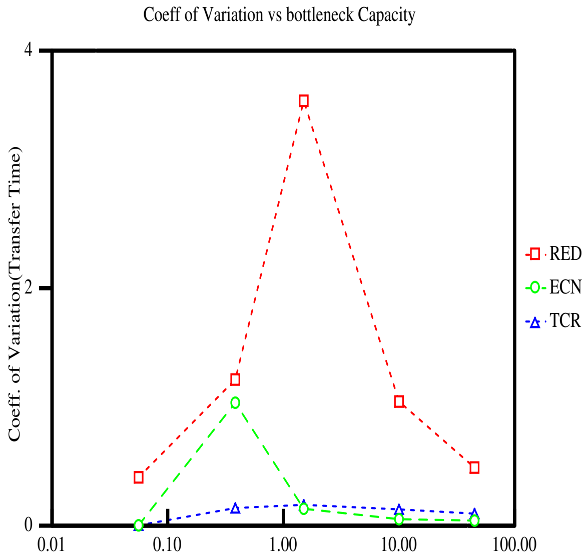

Figure 3: Plot for coefficient of variation in per-flow transfer time - Short transfers

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**# Trans**	μ T. time Milli Secs	**Coeff. of variation (CoV)**
0.056	93.93	401.83/496	15	99624	**0.00282**
0.384	99.07	98.59/174	465	19724	**0.14851**
1.5	99.97	92.00/114	1816	5047	**0.17556**
10	99.85	19.06/94	7054	575	**0.13742**
45	75.76	2.92/59	6974	193	**0.10172**

Table 6: TCP rate control, 100 sources, Short Flows. Result: Lower CoV. Tradeoff: Number of transfers

One important point to note in case of TCP rate control is that even though the transfer time is short , each transfer continues at least for a few RTTs which is long enough to obtain a few rate measurements.

In general, TCP rate control trades off the total number of transfers completed, but consistently reduces the coefficient of variation (CoV) of per-flow transfer times compared to RED. It also has smaller queuing delays compared to ECN. This effect on CoV is also illustrated in Figure 3.

Note that the modeling approximations we have made for evaluation purposes (mentioned in section 1.1.2) affect performance in this case (as seen in reduced utilization and fewer transfers) because the rate allocations translate into non-integral window sizes (in terms of MSS), and the relative changes in window sizes have larger effect on performance since the average TCP window sizes are small during the lifetime of short flows. Packeteer's implementation does not suffer from these modeling approximations.

2.4 Misbehaving Sources

Another key benefit of TCP rate control is in managing misbehaving (non-conforming) TCP sources, which can be easily created by hacking operating systems like Linux. Such sources if unchecked can combine to steal bandwidth from conformant sources (*denial-of-service attacks*), or even lead to congestion collapse. Assuming that such sources respond to ack header fields (receiver window and ack number), TCP rate control can successfully control them and eliminate all their ill-effects on conformant flows. RED and ECN are much less effective in this respect. Therefore rate control can be used as a tool in countering such TCP traffic-based denial of service attacks.

We used a simple technique to create a misbehaving source in our simulation. A normal TCP source starts in the slow start phase and enters the the congestion avoidance phase when the value of *cwnd* crosses *ssthresh*. We disabled the congestion avoidance logic in our code, so that the congestion window *cwnd* always increases exponentially over RTTs. In our simulations, we created a mix of misbehaving and normal sources (50 of each type).

2.4.1 RED : Misbehaving Sources

	Provider Metrics			*User Met rics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**Drops** Packets	**μ Goodput** Mbps	**Coeff. of variation (CoV)**
0.056	94.76	31.85/62	**429**	**0.00009**	**1.88889**
0.384	99.70	28.84/60	**873**	**0.00286**	**1.03846**
1.5	99.66	24.27/64	**1721**	**0.01176**	**1.01190**
10	97.87	18.65/62	**4849**	**0.06109**	**0.50810**
45	96.14	16.58/63	**12334**	**0.21604**	**0.48320**

Table 7: RED: misbehaving sources, homogeneous RTTs, 3000km. Result: Large drop rates, low average goodputs, high CoV

The bold-faced columns in Table 7 indicates that RED performance degrades in terms of number of packets lost, the average per-user goodput and coefficient of variation (CoV). However, provider metrics such as link utilization and queuing delay are optimized. This is another example where *provider metrics being optimized, does not imply that user metrics will be optimized*.

2.4.2 ECN : Misbehaving Sources

	Provider Metrics			*User Met rics*	
Speed Mbps	**Util** Percent	**μ Q/Max Q** Pkts/Pkts	**Drops** Packets	**μ Goodput** Mbps	**Coeff. of variation**
0.056	99.35	**268.97/539**	0	0.00048	**0.45833**
0.384	99.84	**398.91/688**	0	0.00254	**0.25591**
1.5	99.95	**566.71/810**	0	0.01473	**0.06042**
10	99.99	**1554.71/2941**	0	0.09982	**0.03526**
45	100.00	**3249.93/5271**	0	0.44773	**0.05954**

Table 8: ECN: misbehaving sources, homogeneous RTTs, 3000km. Result: Higher Avg Goodput and lower CoV. Tradeoff: high queuing delays.

In this simulation, we have assumed that the misbehaving sources also respond to ECN, but they never use linear increase. Compared to RED, ECN has much better average per-user goodput and reduced CoV because it does not incur the effects of packet loss. However, the effect of misbehaving sources shows up as higher queues (which require larger buffers).

2.4.3 TCP Rate Control : Misbehaving Sources

	Provider Metrics			*User Met rics*	
Speed Mbps	**Util** Percent	μ **Q/Max Q** Pkts/Pkts	**Drops** Packets	μ **Goodput** Mbps	**Coeff. of variation (CoV)**
0.056	99.06	**221.42/413**	0	0.00043	**0.13953**
0.384	99.88	**96.46/200**	0	0.00380	**0.04211**
1.5	99.95	**89.65/200**	0	0.01494	**0.01406**
10	99.99	**27.37/200**	0	0.09982	**0.00311**
45	99.92	**7.28/197**	0	0.44729	**0.00782**

Table 9: TCP Rate Control: misbehaving sources, homogeneous RTTs, 3000km. Result: All metrics optimized. Effect of misbehaving sources eliminated.

TCP rate control clearly stands out in this situation, because there is virtually no effect of misbehaving sources on the performance of the system (eg: compare with Table 3, even though that table uses heterogeneous RTTs). In other words, given that the misbehaving sources respond to receiver window changes in ack headers and modify windows or clock out packets upon receipt of acks, TCP rate control effectively normalizes the performance of such sources.

2.5 Remote Bottlenecks

In all the scenarios studied so far, TCP rate control was implemented at the bottleneck. The interesting question is what happens when TCP rate control is non-local or remote, i.e. not present at the bottleneck of a flow, but elsewhere on its path ? Our primary observation in studying this issue is that since our rate allocation algorithm tracks the current sending rate of a flow, and reduces the allocation if the flow is bottlenecked elsewhere, it limits the overload caused by these flows at the remote bottlenecks. This results in smaller queuing delays even at the remote bottlenecks, though the control is not as effective as when TCP rate control is implemented at the bottleneck. Interestingly, the average goodput and coefficient of variation (CoV) provided by TCP rate-control even remotely is better than offered by RED locally, albeit at the expense of higher queuing delays.

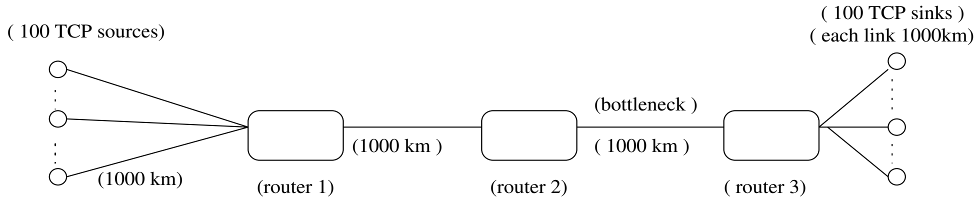

Figure 4: Configuration Template for Remote Bottlenecks

Figure 4 shows the configuration used to study the response of TCP Rate Control when bottlenecks are non-local. Again here we compare performance with RED and ECN. In these comparisons, RED and ECN marking are implemented at the bottleneck (`router 2` in the figure), whereas, TCP rate control is implemented at an upstream, non-bottlenecked node (`router 1` in the figure). Further, in the TCP rate control case, we assume infinite buffers and no buffer management algorithm at the bottleneck.

Tables 10, 11, 12 show the comparative results in this case. We observe that RED again incurs the performance penalties due to packet losses, whereas ECN and TCP rate control do not. TCP rate control has smaller queues than ECN, but the fidelity of control (as seen in the CoV metric) is diminished compared to the cases where it operates locally at the bottleneck.

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	μ **Q/Max Q** Pkts/Pkts	**Drops** Packets	μ **Goodput** Mbps	**Coeff. of variation**
0.056	98.48	30.14/58	367	0.00044	**1.11364**
0.384	99.68	29.00/58	707	0.00288	**0.91319**
1.5	99.70	25.57/59	1651	0.01164	**0.75430**
10	97.54	18.19/56	4850	0.06694	**0.65581**
45	96.15	16.31/50	12150	0.22047	**0.49803**

Table 10: RED: Remote Bottlenecks. Result: Optimized provider metrics. Tradeoff: lower avg goodput, high CoV and packets lost.

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	μ **Q/Max Q** Pkts/Pkts	**Drops** Packets	μ **Goodput** Mbps	**Coeff. of variation**
0.056	99.25	271.45/547	0	0.00048	**0.45833**
0.384	99.82	448.35/731	0	0.00269	**0.20074**
1.5	99.95	531.63/831	0	0.01494	**0.10843**
10	99.99	1343.15/2094	0	0.09982	**0.03967**
45	100.00	2691.98/4194	0	0.44777	**0.01874**

Table 11: ECN: Remote Bottlenecks. Result: Improved avg goodput and CoV compared to RED. Tradeoff: Higher queues

The reason for the performance benefits with rate control even operating remotely is the fact that it tracks bottlenecked flows, and bounds the excess allocation to these flows. As a result, these flows do not overload the remote bottlenecks by a large difference and are also less bursty, leading to shorter queues.

However, we notice that as the bottleneck speed increases (last two rows in Table 12), the remote rate control is less effective because these flows are classified as *non-bottlenecked* or *hungry* for a longer duration. And during this time these flows could overload a remote bottleneck leading to poorer performance.

	Provider Metrics			*User Metrics*	
Speed Mbps	**Util** Percent	μ **Q/Max Q** Pkts/Pkts	**Drops** Packets	μ **Goodput** Mbps	**Coeff. of variation**
0.056	99.25	230.02/440	0	0.00044	**0.25000**
0.384	98.33	108.37/197	0	0.00344	**0.15116**
1.5	99.95	91.33/187	0	0.01494	**0.01740**
10	99.99	181.80/573	0	0.09982	**1.27570**
45	100.00	480.42/859	0	0.44791	**0.58762**

Table 12: TCP Rate Control: Remote Bottlenecks. Result: Improved avg goodput compared to RED. Tradeoff: High queues but still lower than ECN.

3 Related Work

The closest work relative to this paper is proposed by Narvaez and Siu [28], Koike [27], Kalampoukas et al [26] and Satyavolu et al [20]. While these researchers independently suggested the idea of overwriting the acknowledgment fields and pacing the acks (called "acknowledgment bucket" by these authors), this idea was originally invented (patent pending) by Packeteer in 1995-1996, and products based upon these ideas have been shipping ever since [16]. Further, these authors explored such ideas from the point of view of TCP/IP-ATM internetworking, i.e. extending ATM ABR type rate control from ATM edges to TCP end systems.

Narvaez and Siu suggest implementation of a somewhat complex TCP emulation engine at the rate-controller and an acknowledgment bucket. Koike also suggests use of an acknowledgment bucket. Kalampoukas et al develop a new buffer management scheme to directly determine the TCP window, rather than taking a pre-existing rate allocation algorithm and then translating it into a window value. Satyavolu et al suggest a rate-to-window translation scheme based upon another ATM ABR algorithm, ERICA [20]. As mentioned earlier, any good rate allocation algorithm from the ABR literature (eg: see [12]) can be adopted for use in the TCP rate control approach. Moreover, we present more extensive performance analysis compared to these authors, which distinguishes the solution clearly from traditional alternatives.

The TCP rate control approach described in this paper uses a combination of two techniques: a) receiver window modification and b) ack pacing for the rate enforcement process, once max-min fair rate allocations are obtained for competing flows. If only receiver window modification was used, it would lead to burstiness due to abrupt changes in window values. On the other hand, if only ack pacing was used, we would lose control over the sender's window. Since the TCP window is the sum of packets in the forward direction and acks in the reverse direction, ack pacing would successfully control packet queues only when the TCP window is constant. But in cases where the TCP window increased steadily over successive RTTs (eg: slow start), ack pacing alone would fail to quell the queue growth[20].

An alternative rate enforcement technique would be to use per-flow token buckets in the forward direction. The token bucket sizes would vary according to the per-flow rate allocations. This would not be as effective as the method described in this paper because such a scheme would still rely on packet drops to control the TCP window growth and hence would suffer from the adverse performance effects incurred by packet drops.

Several enhancements in TCP have been aimed at reducing the negative effects of packet losses i.e. timeouts and retransmission delays. TCP New-Reno and SACK [29, 14] are the most prominent ones which are also being deployed in server-side upgrades. We chose TCP Reno in this simulation study because of the overwhelming majority of the installed base. We would expect the performance gap seen in this paper to reduce if either TCP New-Reno or SACK are used in end systems, but we expect rate control would still be useful in these cases because of the use of explicit, detailed feedback, and in cases when non-conforming flows are present.

TCP rate control is sometimes viewed as a misnomer because TCP does not use a rate parameter or leaky bucket to regulate packets. But if TCP's window remains constant (equal to the receiver window), the rate of packets clocked out by TCP (in packets/s) is equal to the rate of acknowledgments. Even if the window varies, the rate allocation algorithm (step 6 in algorithm 1) quickly tracks this value and regulates TCP's rate implicitly by regulating the rate of acks.

TCP rate control is different compared to scheduling schemes like WFQ, priority queuing, CBQ [5, 30] because rate control operates on a single FIFO queue (with per-flow information), whereas scheduling schemes use multiple queues. In fact, TCP rate control can be applied independently to each queue in these scheduling algorithms, especially when many flows may share any given queue. The interesting aspect about the underlying rate allocation algorithm is that, it can potentially achieve max-min fairness even with a FIFO queue (but requires closed loop rate feedback), whereas scheduling algorithms can achieve max-min fairness by maintaining per-flow queues in an open loop manner. Further TCP rate control is orthogonal to the diff-serv approach and can be used to complement diff-serv in potentially many ways which in itself is a topic of further research. From the complexity point of view, the space complexity for TCP rate control is O(N) as it stores per-flow information and the time complexity for processing a packet or an ack is

O(1). The rate allocation algorithm complexity can vary from O(1) to O(N) depending upon the particular algorithm chosen [12]. As a useful data point regarding scalability, we note that Packeteer products handling over 20,000 flows simultaneously and running at 45 Mbps have been deployed for over a year.

TCP rate control is different from buffer management algorithms like RED and FRED [7, 13, 15] in that the latter assign non-zero packet loss probabilities p_i and queuing delays (which affect RTT_i), whereas rate control assigns a zero loss probability and uses the receiver window (W_{rcvr}) and rate of acks as the primary control mechanisms. Hence, as argued in this paper, it side-steps the effects due to packet loss. However, we note that TCP rate control is TCP-friendly [17] in the sense that when packet loss does occur and congestion window drops below the assigned window value, the rate controller never artificially inflates it.

4 Deployment Issues

RED is stateless and does not depend upon any protocol headers for its functionality. Therefore it can be deployed at any queue in the network. It is also simple and has O(1) operation. It has therefore been widely recommended as an active queue management technique, though no standardization process is required [3].

ECN requires support from both end systems (host TCP/IP stacks) and routers. In other words, it requires some minimal standardization and has therefore been defined as an experimental RFC [19]. Since it would take a long time to upgrade and deploy both key routers and hosts, ECN has the longest deployment cycle among the three alternatives.

TCP rate control is stateful (maintains state per TCP flow) and depends upon ability to read and write to TCP headers of acknowledgments. TCP rate control works best when both the TCP packet and ack flows are accessible to it. However it can perform the core functions even under limited asymmetry (eg: ack flow alone accessible). Since it requires reading and writing into TCP headers, these headers should be accessible (eg: not encrypted or authenticated). However, given these conditions, it can function transparently and does not require any standardization or support from existing routers or end systems (hosts). Though it maintains per-flow state, through information compression and O(1) computation, reasonably scalable commercial implementations (over 20,000 simultaneous flows, 45 Mbps) have been deployed which is sufficient for network edges and enterprise networks. Due to the above constraints, the solution is more applicable to network edges (ISPs and enterprises) rather than core nodes, given that the TCP headers are accessible.

5 Summary and Future work

TCP rate control is a new technique which transparently augments TCP performance through indirect control of its rate (achieved by manipulating the ack stream only). Performance-wise, it avoids the adverse performance effects due to packet losses. Further, it can provide protection against misbehaving (non-conformant) TCP sources and improve performance even if the bottlenecks are non-local. Though it is somewhat constrained in its deployment space compared to stateless algorithms like RED, it has been quickly and effectively applied at network edges of ISPs and enterprise networks.

Future work will focus on further augmenting performance in the presence of remote bottlenecks using diff-serv marking, using more accurate modeling and benchmarking performance against TCP-SACK/New-Reno.

References

[1] H. Balakrishnan, V. Padmanabhan, S. Seshan, M. Stemm and R. H. Katz, "TCP Behavior of a Busy Internet Server: Analysis and Improvements," *Proceedings of IEEE Infocomm*, San Francisco, CA, USA, March 1998.

http://www.cs.berkeley.edu/ hari/papers/infocom98.ps.gz

[2] J. Bolot and A. Vega-Garcia, "Control Mechanisms for packet audio in the Internet," *Proceedings of IEEE Infocom'96,* 1996.

[3] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, "Recommendations on Queue Management and Congestion Avoidance in the Internet," *Internet RFC 2309*, April 1998.

[4] A. Charny "An Algorithm for Rate Allocation in a Packet-Switching Network with feedback", Masters thesis. MIT 1994

[5] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queuing Algorithm," *Internetworking: Research and Experience*, Vol. 1, 1990, pp. 3-26.

[6] W. Feng, D. Kandlur, D. Saha, K. Shin, "Techniques for Eliminating Packet Loss in Congested TCP/IP Networks," *U. Michigan CSE-TR-349-97,* November 1997.

[7] S. Floyd, and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance," *IEEE/ACM Transactions on Networking*, Vol. 1, No. 4, August 1993, pp.397-413.

[8] S. Floyd, "TCP and Explicit Congestion Notification", *ACM Computer Communication Review*, Vol. 24, No. 5, October 1994, pp. 10-23. ftp://ftp.ee.lbl.gov/papers/tcp_ecn.4.ps.Z

[9] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of the SIGCOMM'88 Symposium*, pp. 314-32, August 1988.

[10] R. Jain, "The Art of Computer Systems Performance Analysis," *John Wiley & Sons Inc.,* 1991.

[11] S. Kalyanaraman, R. Jain, R. Goyal, S. Fahmy, and S.C. Kim, "Use-it or Lose-it Policies for the Available Bit Rate (ABR) Service in ATM Networks," Computer Networks and ISDN Systems, Vol. 30, No. 24, 14 December 1998, pp. 2293-2308

[12] S. Kalyanaraman, "Traffic Management for the Available Bit Rate (ABR) Service in Asynchronous Transfer Mode (ATM) networks" *Ph.D. Dissertation*, Dept. of Computer and Information Sciences, The Ohio State University, August 1997.

[13] D. Lin and R. Morris, "Dynamics of Random Early Detection," *Proceedings of SIGCOMM'97*, August 1997.

[14] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgement Options," *Internet RFC 2018*, October 1996.

[15] R. Morris, "TCP Behavior with Many Flows",*IEEE International Conference on Network Protocols*, October 1997.

[16] Packeteer Inc., http://www.packeteer.com/

[17] J. Padhye, V. Firoiu, D. Towsley, and Jim Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation," *Proceedings of SIGCOMM'98*, Vancouver, August 1998.

[18] K.K. Ramakrishnan and R. Jain, "A Binary Feedback Scheme for Congestion Avoidance in Computer Networks with Connectionless Network Layer," *Proceedings of SIGCOMM'88*, August 1988, pp. 303-313.

[19] K.K. Ramakrishnan, S. Floyd, "A proposal to add Explicit Congestion Notification (ECN) to IPv6 and to TCP," *IETF Internet Draft*, November 1997, Available as http://ds.internic.net/internet-drafts/draft-kksjf-ecn-00.txt

[20] R. Satyavolu, K. Duvedi, S. Kalyanaraman, "Explicit rate control of TCP applications," *Unpublished Manuscript* 1999. http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers/iwqos99-rate.ps

[21] W. R. Stevens, "TCP/IP Illustrated, V. 1," *Addison-Wesley, Reading, MA*, 1995.

[22] W. R. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," *Internet RFC 2001*, January 1997.

[23] ATM Forum Traffic Management, "The ATM Forum Traffic Management Specification Version 4.0," April 1996. Available as ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.ps

[24] S. Kalyanaraman, R. Jain, R. Goyal, S. Fahmy, and R. Viswanathan, "The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks" *IEEE Transactions on Networking*, to appear, 1999. Available from: http://www.cis.ohio-state.edu/ jain/papers/erica.htm.

[25] S. Karandikar, S. Kalyanaraman, P Bagal and B. Packer. "TCP Rate Control for Congestion Avoidance" *Technical Report*, October 1999. Available from http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers-rpi.html.

[26] L. Kalampoukas, A. Varma, K.K. Ramakrishnan, "Explicit Window Adaptation: A Method to Enhance TCP Performance," *Proceedings of INFOCOM'98*, April 1998.

[27] A. Koike, "TCP flow control with ACR information," *ATM Forum/97-0998*, December 1997.

[28] P. Narvaez and K.Y. Siu, "An Acknowledgment Bucket Scheme for Regulating TCP Flow over ATM," to appear in *Computer Networks and ISDN Systems Special issue on ATM Traffic Management*, 1998.

[29] Floyd, S., and Henderson, T., "The NewReno Modification to TCP's Fast Recovery Algorithm", *Internet RFC 2582, Experimental*, April 1999.

[30] Floyd, S., and Jacobson, V., "Link-sharing and Resource Management Models for Packet Networks"*IEEE/ACM Transactions on Networking, Vol. 3 No. 4, pp. 365-386*, August 1995. Available from: ee.lbl.gov/nrg-papers.html