Techniques for Explicit Feedback Control of TCP [1]

Ramakrishna Satyavolu[2], Ketan Duvedi, Shivkumar Kalyanaraman, Prasad Bagal
Department of Electrical Computer and Systems Engineering,
Rensselaer Polytechnic Institute,
110, 8th Street, Troy, NY 12180-3590
Phone: 518-276-8979, Fax: 518-276-2433, Email: shivkuma@ecse.rpi.edu;
Internet: http://www.ecse.rpi.edu/Homepages/shivkuma

Abstract

Two important issues in TCP-based flow control are fairness and control of end-to-end delay. This paper examines different methods of controlling the rate and window of TCP to achieve these goals. The key idea is to use existing rate-control algorithms to calculate rate feedback and use new techniques to control TCP sources using the feedback calculated. Specifically we propose two rate-to-window translation schemes for explicit window feedback to TCP and one variant of an acknowledgment bucket scheme (which controls TCP rate by controlling the rate of acknowledgements). These techniques appear to be well suited for application in ATM edge devices or in enterprise routers. We study the dynamics of TCP traffic under different methods of rate control and show through simulations that our techniques can considerably improve fairness, throughput and end-to-end delay properties of TCP applications. We have worked with Packeteer Inc. which makes a successful product using similar techniques which scales to over 20,000 simultaneous flows, and has been deployed in edges of enterprise and ISP networks.

1 Introduction

There has been considerable interest in studying the dynamics of legacy TCP/IP applications over Asynchronous Transfer Mode (ATM) (see [1, 2, 3, 4, 5] and references therein). These studies have indicated that an Available Bit Rate (ABR) service implementation can provide low-delay, (max-min) fairness, and high throughput inside the ATM network for ftp-like applications and a reasonable tradeoff for bursty or short-lived applications like HTTP [3]. However it is observed that TCP/IP queues build up at the edge of the ATM network. The reason for this behavior is that the explicit rate control algorithms within the ATM cloud push the queues to the edges and maintain a low-delay, high-throughput performance inside the cloud. In contrast, the Unspecified Bit Rate (UBR) [14] and Guaranteed Frame Rate (GFR) service implementations allow traffic into the ATM network, and if buffering is insufficient, tackle the issues of throughput, and fairness through intelligent buffering, scheduling and drop policies. If the effects of fragmentation are ignored, these latter service classes resemble the best-effort service obtained from Internet routers today.

Given that we know how to control steady state queue delay and achieve high throughput in ATM ABR implementations, the natural question which arises is how one can provide the benefits of ABR end-to-end, i.e., provide better control of delay, throughput and max-min fairness of TCP connections. An innovative approach suggested recently [4, 5] involves maintaining an

[1] This is an unpublished technical report, an expanded version of an earlier ATM Forum contribution. This work was sponsored in part by NSF Contract # ANI9806660, DARPA Contract # F30602-97-C-0274, and Packeteer Inc.

[2] Ramkrishna and Ketan are now with Microsoft Corp

"acknowledgment bucket" (also called "ack-bucket") at the edge device (or router) and releasing the acknowledgments (which in turn controls the rate of TCP) based upon the Allowed Cell Rate (ACR, an ABR parameter) of the VC carrying the TCP connection. The acknowledgement bucket need not explicitly store each acknowledgement; due to the cumulative nature of acks, just the latest information needs to be stored per connection. If packet queues in the forward direction can be traded for long ack queue which can be represented in O(1) state, this is a fine tradeoff which can be made.

In this paper, we first study a more direct mechanism to control TCP sources, which is to control its maximum window size based upon the explicit rate calculated by the ERICA+ algorithm [16]. As described later in this paper, this mechanism can be used independently or in conjunction with the "acknowledgment bucket" mechanism. In this paper we study the feasibility and benefits of this mechanism (in terms of throughput, delay and fairness), and its impact on TCP traffic dynamics. Though we were unaware of it at the time of doing this research (mid 1997), a similar approach has been used in successful enterprise bandwidth management products by Packeteer [18] and has also been recently proposed in an attractive scheme by Kalampokulas, Varma and Ramakrishnan [17].

Our approach differs from Packeteer in how we arrive at the TCP window value - Packeteer derives the window values based upon policy-based threshold settings as well as a dynamic rate control algorithm (different from ours). Our work differs from Kalampoukas, Varma and Ramakrishnan's approach in that we take an existing rate-control scheme (any existing scheme can be used) and do a rate-to-window translation (thus inheriting its fairness, stability and robustness properties), whereas they develop a new scheme based upon buffer occupancy. Their work also shows that explicit window control is more effective than buffer management techniques like Random Early Discard (RED) [13] due to its inherence congestion (and loss) avoidance approach.

We then identify a variant of the acknowledgement bucket idea whereby the acks are simply clocked out at the rate calculated by our rate-control algorithm, ERICA+. References [4, 5] also use a rate to control the ack bucket. The ack-bucket algorithm used in [4] attempts to model the TCP source's behavior in the router, and use the model to decide when to release the acknowledgements. This method may be restricted by the accuracy and complexity of the TCP source model used. Reference [5] uses the VC's ACR at an ATM end-system, and a simple queue-threshold based scheme to decide when to release the acknowledgements. The queue-threshold based scheme is used to compensate for the end-system queues which may accumulate. The fairness of the scheme is not clear and the choice of the thresholds should be done carefully to avoid buffer overflows. Our approach is different in that it simply clocks out the acks at the explicit rate calculated by the ERICA+ scheme [3, 16]. The ERICA+ scheme controls the queueing delay and calculates a fair allocation for the contending sources. The ack bucket simply enforces the fair allocations. We then study the combination of this scheme with one of our proposed TCP explicit feedback mechanisms to achieve multiple tradeoff points.

We also study the dynamics of TCP under explicit rate or window control. We conclude that considerable care has to be taken in controlling TCP windows or rates remotely. Specifically we show that a) window corrections can lead to additional burstiness in network traffic, b) simplistic rate-to-window translations could be useful in controlling delay, but need not necessarily be fair or efficient, and c) simplistic ack-rate control may not lead to fairness or queueing delay control.

This paper represents our initial work (done in 1997) in the area. We have subsequently worked with Packeteer, Inc. (www.packeteer.com) which makes a TCP rate control product based upon explicit control of the TCP window (using the ack number and receiver sequence number fields in acks) and acknowledgement shaping (to achieve smoothness). In a recent detailed performance study [23] we compare their algorithm to RED and ECN. Though the technique maintains per-flow state, with the use of information compression (in the ack bucket as explained earlier), the storage and state requirements of the implementation scale very well (Packeteer products supporting over 20,000 simultaneous flows are now in operation). Since the method is transparent to end-systems or routers, it has been deployed easily, extensively and effectively. Due to requirements of symmetry (or limited asymmetry), and required access to transport level headers (not available if encrypted or authenticated), the solution is more applicable to network edges rather than cores.

2 TCP congestion control evolution

The TCP congestion control algorithm [6] is used to provide a reliable end-to-end transport service. The TCP congestion control protocol is extremely robust and has been shown to work under pathological conditions. However it has certain well-known drawbacks relating to its performance, some of which are enumerated below:

- Unfairness (a connection which sends more gets a larger window) [13]
- Little control over end-to-end delay characteristics [20]
- Late congestion detection (due to the implicit packet-loss-based detection technique)
- Degraded throughput due to bursty loss of packets (due to triggering of the long TCP timeout) [8, 9]
- Traffic synchronization effects [13]
- Burstiness of traffic depending upon reverse-path congestion [20]
- Long ramp up times in long-delay bandwidth paths [21]

The Internet Engineering Task Force (IETF) and independent researchers have proposed several improvements to TCP/IP-based control at the transport and network layers. Proposed transport layer enhancements include the fast retransmit and recovery (FRR) algorithms [7], selective acknowledgments (SACK) [8] and Explicit Congestion Notification (ECN, one bit explicit feedback) [9]. The former enhancements (FRR, SACK) aim at improved throughput, timeout avoidance and quick recovery from burst losses, while ECN aims to improve fairness and delay characteristics experienced by connections. Recently, proposals to improve the startup dynamics of TCP (esp. for satellite links) have been seen [10]. Network layer enhancements are aimed at improving fairness and throughput. These include mechanisms like scheduling [11] and packet discard policies [12, 13].

These enhancements have been found to improve TCP throughput, avoid the expensive TCP timeout, and react well during burst loss of packets. Most of these proposals seek to achieve greater efficiency to attacking the problems associated with TCP retransmission, avoiding the time delays caused by slow start and timeout. However, providing some form of fairness and delay control still remain fairly open problems, though the ECN [9] proposal seeks to partially address it. Delay and fairness are two important user-perceived metrics in WWW performance which would justify interest in these metrics. In this paper we study techniques which address these issues.

3 Explicit Control of TCP rate

Given a rate r calculated by a max-min fair scheme at a switch/router/atm end system, our goal is to make TCP send data at rate r. Our approach is to convert r into a window value W by multiplying it by a time quantity representative of the round trip time, T.

W = r*T

We will see later (section 4.1.1) that there are several possible ways of choosing T and the method is sensitive to this parameter. Specifically, a small T can result in underutilization; a large T in queueing; and a constant T for different connections in unfairness. We will explore these aspects in simulation later in the paper. The rate r can be calculated using a switch algorithm like ERICA+ [16], or can be gotten by simply using the Allowed Cell Rate (ACR) parameter of an ABR VC. 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 [15]:

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 needs to adjust the 16-bit window feedback, W, apropriately (eg: right shifting by the factor). Further, the TCP checksum needs to be adjusted as follows:

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

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

Note that the header modification does not violate the TCP protocol and as a result requires no standardization. Also note that even if the TCP packet is fragmented across several IP datagrams, the window update and checksum adjustment needs to be made only in those fragments where the TCP header is available. Also this adjustment must be applied to every acknowledgment seen (including piggybacked acknowledgments), though the rate, r (if it is computed at the node) need not be computed on a per-packet basis.

Though we maintain some per-flow state in our method of rate computation, one could use any of the existing rate-control algorithms in the literature which optimize on the cost metrics of interest (for a survey, please refer [3]). The key benifit of choosing a well-known algorithm is to inherit its control and performance properties such as stability, robustness, parameter sensitivity, throughput, max-min fairness and queueing delay control while concentrating the new effort on carefully building the translation mechanisms from rate-to-window control. It may also be possible for routers/edge devices to control several TCP flows as a single unit (for example, all flows from some particular domain can be rate controlled as a single unit, albeit at the loss of accuracy of control).

However, we do observe that these techniques may not have applicability in environments such as ISP core network interiors where issues such as encryption/authentication, route flaps and increased packet processing due to large number of flows/packets arise. Encryption/authentication rules out any possibility of writing to the TCP header, while route flaps (due to instability in routing algorithms) can result in undesirable oscillations. But with options such as Explicit Congestion Notification (ECN) and TCP-aware buffer management techniques becoming available at the core, it may become feasible to use rate-control techniques indirectly through these options. This is an area for future study. However, as mentioned earlier, this concept has been successfully productized by Packeteer Inc. (www.packeteer.com) and deployed primarily in enterprise and ISP network edges.

4 TCP Dynamics with Rate Control

TCP uses window control to send packets into the network. In general if X(t) packets are sent into the network in time interval t, the rate of the connection is dX(t)/dt. In the case of window control with a fixed size window, W, the rate of the connection in the steady state is equal to the rate of receipt of acknowledgments (acks). In the case of TCP, the window size increases by either 1 (or 1/W(t) during congestion avoidance) for the receipt of every non-duplicate ack. As a result, the rate of the TCP connection in every round trip time (RTT) is either 2*W/T (W/T + 1) in every round trip time T, where W is the window size at the end of the previous RTT. At the end of the current RTT, the new window size is 2*W (or W+1).

In other words, the TCP rate in the worst case may grow exponentially in successive RTTs. Also note that, when the TCP source is in its startup phase, its window size is small, and as a result does not receive acknowledgments regularly - they are received in bursts. The rate of TCP as calculated above is the average rate over a period of RTT. The startup TCP rate measured over small intervals (e.g. at the router) could be much higher (during the burst) or lower (during idle periods) but should equal the average rate once the acknowledgments are received regularly (the "pipe" is filled). Once the pipe is filled, the ack rate could still be irregular depending upon the queueing dynamics inside the network, and use of delay ack timer at the destination. However, for our discussion, we shall not involve the effect of these latter factors.

In the following sections we will considering two ways of controlling the TCP rate: a) by limiting the maximum window size of the TCP source, and b) by controlling the rate of acknowledgments (without explicitly writing to the window field). We will also consider combinations of these two methods.

4.1 TCP Dynamics with Explicit Feedback Control

Firstly, we recall that our the window adjustment technique described in section 3 controls only the maximum receiver window (Wr), which acts as an upper bound on the TCP congestion window (CWND) variable at the source. The method would not have any effect if a) CWND < Wr, or b) Wr < W (feedback value larger than receiver's max window itself). Note that while case b) is a permanent condition where the feedback calculated is ineffective, case a) can be assumed to be a transient condition.

Specifically, in case a), CWND is initially small (slow-start increase). But, once CWND > Wr, the window available to the source (which we call "actual window") is Wr. The difference between the congestion window and actual window is never sent into the network (Figure 1). As a result all connections experience reduced queueing delays. Thus we have the basis of a scheme that reduces queueing latencies by controlling the TCP transmitters directly, and remotely.

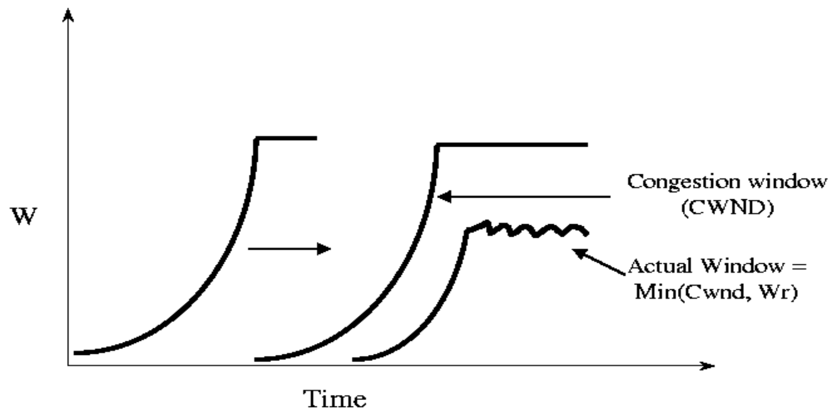


Figure 1: Congestion Window vs Actual Window

Another issue with explicit window control is burstiness. The control of TCP window does not immediately control the TCP rate and make it smooth. Specifically, as illustrated in Figure 2(a), sudden increases in Wr (due to fluctuations in the available rate r) may result in a sudden burst of packets in the network. Similarly a sudden decrease in Wr may result in an extended idle period (because the source is waiting for the acknowledgments for excess packets sent when the window was higher). Even if Wr is constant (r is in the steady state), the rate of packets sent into the network depends upon the rate at which the acknowledgments are received (Figure 2(b)). The rate of acknowledgments depends upon factors such as the TCP source rate, the queue lengths (which may fluctuate), and delay ack timer at the TCP destination (we ignore ack bucket control for this discussion). As a result the effective rate of TCP depends both upon the changes in the maximum receiver window Wr (affected by our scheme), and the rate of acknowledgments received (dependent upon queueing dynamics, also partially dependent upon our scheme). We now develop two methods of controlling the receiver window Wr based upon a rate r.

4.1.1 Two methods for Translating Rate Feedback into Window Feedback

In this section, we consider two ways of choosing T, the time value used to translate r to W. One approach is to treat the advertised rate value as an average allocation over the period of the flow's round-trip delay, T_i, and calculate the window as the product of the rate, r, and the flow's round trip delay, T_i. The second approach is advertise a window value based upon a single time estimate T.

The first approach, called "translation using individual source RTT" requires the round trip delays of individual sources. The window value given as feedback indicates the ideal delay-bandwidth product per connection, where the delay is T_i, and the bandwidth is r. Once the sources' congestion window values reach r*T_i, and the flow of acks back to the source is regular,

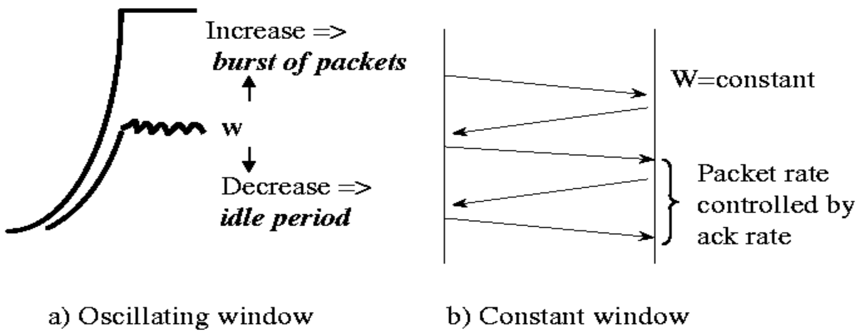


Figure 2: Burstiness due to explicit window control of TCP

the allocation is fair and efficient. This is illustrated in figure 3(a). Note however that the rate fluctuations in r, would result in magnified window changes in larger RTT connections. So, as described in section 5 we use a conservative increase technique to avoid such burstiness. We show using simulations in section 6 that this scheme can achieve fair and efficient allocations, with minimal buffer requirements and queueing delays. As a final point, note that one way of determining an upper bound on the round trip time of a connection is to send an ICMP echo message to both source and destination.

The second approach, called "translation using a fixed-RTT," is to advertise a single window value based upon a time estimate T. The simplest approach is to choose a constant T. Our simulations in section 6 show that while this method can control delay to small values, it does not eliminate unfairness. However its worst case performance in terms of fairness is better than vanilla TCP in that it allows every connection to get a non-trivial share of the bandwidth (though not necessarily the fair share). Also if T is chosen to be too small or too large, it would lead to reduced utilization or increased queueing delay respectively. We have done simulations to illustrate these latter aspects, but for brevity do not include them here.

The reason for the unfairness in the fixed-RTT translation is explained as follows. We are choosing a single value T to translate the rate feedback for heterogeneous RTT connections. If T $<$ Max RTT, the largest RTT source i is limited to a window of r*T which is less than r* (Max RTT). As a result, source i cannot send at a rate r continuously since the acks cannot be sustained at a rate r over a round trip time of Max RTT. This is illustrated in figure 3(b). If T $\geq$ Max RTT, the above problem can be overcome, i.e., it is possible to achieve a state where the largest RTT connection i finally gets acks back at a rate r. However, the distribution can still be unfair because of the following situation. All sources now get a fixed window size r*T, but a smaller RTT connection j could transmit its entire window (r*T) within its RTT (RTT_j), i.e. it sends traffic at a rate r*T/RTT_j. However, the largest RTT connection i in this state can achieve a maximum rate of only r*T/RTT_i = r*T/(Max RTT), which is smaller than the

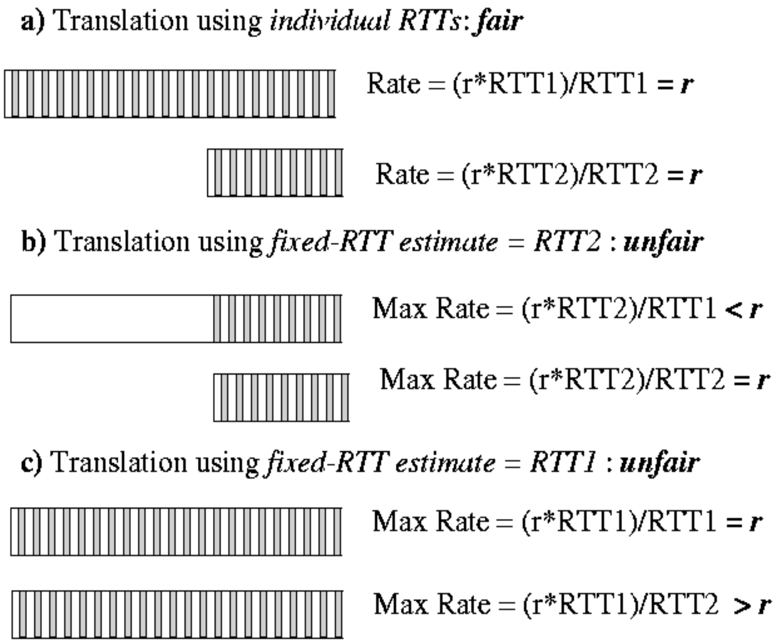


Figure 3: Fairness issues with explicit window control of TCP

rate of connection i, an unfair situation. This is illustrated in figure 3(c).

We note that for this latter case i.e., T $\geq$ Max RTT, a simple ack bucket scheme can be used in conjunction with the explicit feedback scheme to achieve fairness. Specifically, consider an ack bucket scheme enforces the rate of acks to be equal to r which augments the feedback scheme and the window values of sources Wi, Wj are greater than r*Max RTT. We now have a condition where the ack bucket scheme is working with fixed window sources, and regular ack flow. As explained later in section 4.2, this can lead to a fair distribution of rates. As a final point, note that the buffer requirements for this method (Buffer = Sum (r*Max T_i)) will be larger than that for the first method, where we use individual RTT values T_i or feedback calculation (Buffer = Sum (r*T_i)). Now the fairness achieved by using the acknowledgement bucket in addition to explicit feedback leads us to investigate the former in its own right.

4.2 TCP Dynamics with Ack Bucket Control

Recall that the ack-bucket idea is to control the rate of TCP packets in the forward direction by controlling the rate of acknowledgements in the reverse direction. The TCP dynamics under ack-bucket control can be understood by using a simple ack-bucket which, when given a rate r, releases ("clocks out") the acks at a rate r. The first effect of the ack-bucket control is that the rate of increase of the congestion window is slowed down (Figure 4). Further, the packet queue in the forward direction is reduced at the expense of the ack-queue in the reverse direction (the window is the sum of the packets in the forward direction and the acks in the reverse direction). It has been shown [4] that the ack-queue can be implemented efficiently, even when duplicate or piggybacked acks are seen. Moreover, the "smoothness" or "burstiness" of TCP transmission can be controlled by the ack-rate.

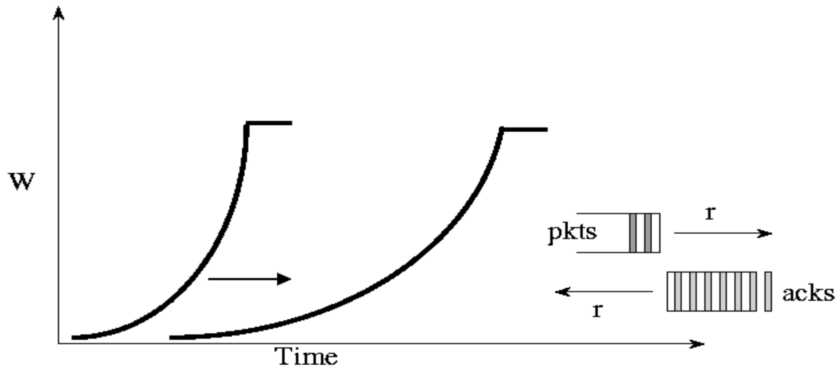


Figure 4: Effect of acknowledgement bucket control

We show that a careful control of the ack-rate can also lead to a fair distribution of rates. Let us first examine an example of unfairness. Assume that there are two connections passing through the router which each get a rate r, and that one connection has a small RTT (RTT1) and the other a large RTT (RTT2). Further, assume that both connections currently have the same window size W such that r*RTT2 $\gg$ W $\gg$ r*RTT1. In this state, connection 2 is idle most of the time, whereas connection 1 is receiving acks continuously. If both connections are in the exponential increase phase, connection 1 would increase its window at rate r continuously, whereas connection 2 would increase its window at a rate r for a period W/r for every round trip (RTT2). Hence connection 1 sends more packets through than connection 2 during this transient phase (temporary unfairness).

Now once the long RTT pipe is filled (i.e. connection 2's window grows to r*RTT2), it gets acks regularly and increases its window at a rate r (like connection 1). Since both connections are increasing their rate of transmission exponentially, the queue increases at rate 2r. An additional mechanism is necessary to quell the queue growth. *In general, controlling the rate of acks (in the reverse direction) to the output rate r of the packet queue in the forward direction is insufficient because it does not take into account the packet queue building up in the forward direction.* A separate algorithm is necessary to take that into account.

Reference [5] proposes a queue threshold based algorithm to control the ack release rate (basically a simple ABR switch congestion control algorithm) which reacts upon queue length crossing a threshold by cutting down the rate of acks by a constant factor, H. Observe from the above discussion that it is necessary that H $\geq$ 2 to guarantee stop of queue growth in the above scenario. In general, the correct rate under such conditions can be better calculated by a explicit rate switch algorithm like ERICA+ [16]. Hence, our proposal is to use a simple ack-bucket, where the acks are clocked out at the rate calculated by an explicit rate algorithm like ERICA+. We demonstrate in section 6 that this method can result in fair allocations.

Observe, however, that the simple ack-bucket scheme (without ERICA+) can achieve fairness *if the TCP source windows are kept constant and the window of source i, Wi, is no less than r*RTT(i).* Under these conditions, acks return continuously to both sources (irrespective of RTT) at rate r, and since the window Wi is constant, each ack simply clocks a new packet into the network. In other words, the rate of both connections is now r, a fair distribution. The ack bucket scheme would also be near-fair in the above situation if the TCP sources are in their congestion avoidance phase where the window increases slowly. One of our proposals here involves showing how a simple explicit feedback scheme (the fixed-RTT method described in section 4.1.1) can augment the ack-bucket scheme to maintain a constant window at TCP sources (and as a result achieve fairness).

5 Simplified ERICA+ algorithm for Explicit Rate Calculation

We use the ERICA+ algorithm as described in [16, 3] for our simulations. We note that any rate-based algorithm studied for ABR can be used in place of ERICA+ in such an implementation. Also the rate-control we describe need not be implemented in all routers in a domain. As Packeteer has demonstrated with its business success, deploying such algorithms in the critical bottlenecks is sufficient to improve performance dramatically.

ERICA+ is a well-known algorithm in the ATM Traffic Management community, which achieves the goals of fairness, high utilization, and low delay using rate-based control. Briefly, the algorithm measures the input rate, number of active flows, available capacity, the current queueing delay, and the current rate of each flow and combines these metrics to determine the feedback to sources (new rates the flows should send at). The algorithm is sensitive to some key parameter settings, especially the interval chosen to measure the metrics. Simulation and analysis of the algorithm is found in [16, 3].

The key change we make to the algorithm is to limit rate increases to 10% of its previous value. This step, which is similar to the limit imposed by the ABR source Rate Increase Factor (RIF) parameter, is done to prevent burstiness which can result due to sudden window changes at the source (see section 4.1). Rate decreases are not limited. The source rate of each connection is measured at the end of an averaging interval and the new feedback rate is calculated at that time. *All of the calculations are done at the end of the averaging interval; no rate updates are made in the middle of averaging intervals.* The resulting algorithm is:

<u>Initialization:</u>

FairShare = 0;
MaxAllocPrevious = MaxAllocCurrent = FairShare;

<u>End of Interval:</u>

Measure Total Available Capacity
Measure Input Rate
Target Capacity ← f(Q)* Total Available Capacity
z ← Input Rate/Target Capacity
Measure number of active sources
FairShare = Target Capacity/Number of active sources
FOR each session i DO
    Rate[i] = Num Packets In Interval [i]/Interval Length

VCShare ← Rate[i]/z
IF (z > 1+ delta) ER[i] = Min(Max(VCShare, FairShare), 1.1*ER[i])
ELSE ER[i] = Min(Max (MaxAllocPrevious, VCShare), 1.1*ER[i])
MaxAllocCurrent = Max (MaxAllocCurrent,ER[i])
IF (ER[i] > FairShare AND Rate[i] < FairShare)
THEN ER ← FairShare
ENDFOR
MaxAllocPrevious = MaxAllocCurrent
MaxAllocCurrent = FairShare

TCP Acknowledgement Processing:

Wr_{new} ← Min(Wr_{old}, ER[i]*T[i]) // T[i] is either a fixed common value or a per-connection value
Delta = $Wr_{new} - Wr_{old}$ // one's complement subtraction
TCP_checksum = TCP_checksum + Delta //one's complement addition

6 Simulation Results

The simulations were run using the following parameters from the suggested canonical range in [3]. Details of parameter issues are presented in that reference.

Parameter	Value
Simulation time	5 s
Switch Avging Interval (ERICA+)	5 ms
Target Queueing Delay (ERICA+: T0)	1.5 ms
Queue Drain Limit Factor (ERICA+: QDLF)	0.5
a (ERICA+)	1.15
b (ERICA+)	1
Link speed	155.52 Mbps
Maximum Segment Size (TCP: MSS)	1024 bytes
SSThresh (TCP)	600000 bytes

We use an infinite buffer in our simulations, to allow us to assess the maximum queue size (and queueing delay) possible with the different options. As a practical matter, we note that most packet drops in networks occur either at the egress or ingress queueing points of enterprise network sites. So, by careful control of precisely these queueing points with reasonable per-flow buffering and flow isolation, we can achieve lossless, low-delay and fair performance with TCP rate control. Unbounded queues do not arise because windows can be explicitly throttled and excess packets do not leave the TCP source. This is why we have not considered cases with small buffers and packet losses - the strength of this approach is avoid the complex dynamics that arise due to packet losses. We have illustrated the effect of such dynamics in a recent and more detailed comparative performance study [23].

In these simulations, we use queue length as a proxy for end-to-end delay because it is the variable component of delay and in some cases is the dominant factor. We also present a sample simulation with the random early detection (RED) buffer management scheme as a point of comparison in section 7. We have not done extensive simulations with HTTP traffic, but Packeteer's products which follow a similar approach have worked well under such situations as well, and support over 20,000 simultaneous flows [18]. As mentioned above, while the simulations are limited, we have performed a detailed comparative simulation study and we have operational experience from thousands of Packeteer rate-control products which we believe reasonably validates this approach [23]

6.1 Simulation Results with a Heterogeneous RTT Configuration

We use the configuration shown in Figure 5 for studying heterogeneous RTT cases. The TCP sources named STCP #1..15 are the sources for packets and the TCP connection objects names DTCP #1..15 are the corresponding sinks for the TCP sources. There are three groups of 5 sources each. The first group has access links of 3 km each (or an RTT of 2012 km = 10 ms). The second group has access links of 300 km each (or an RTT of 3200km = 16ms). The third group has access links of 3000km (or an RTT of 14000 km = 70 ms).

While this simulation configuration appears simplistic, it is similar to most of the deployment scenarios of this approach. Enterprise sites have 10-100 Mbps Ethernet internally and interconnect sites with a low speed T-1 (1.5 Mbps) or 56 kbps connectivity. ISP networks are also hierarchically organized with high speed backbones and low speed ingress/egress links. So, the bottlenecks are usually at the ingress/egress network edge points of ISPs or enterprises (which is where this method is most applicable). These core is currently overprovisioned in both cases. While we look at a high speed bottleneck (155 Mbps) in this simulation set, our parallel study looks at a full range of link speeds [23].

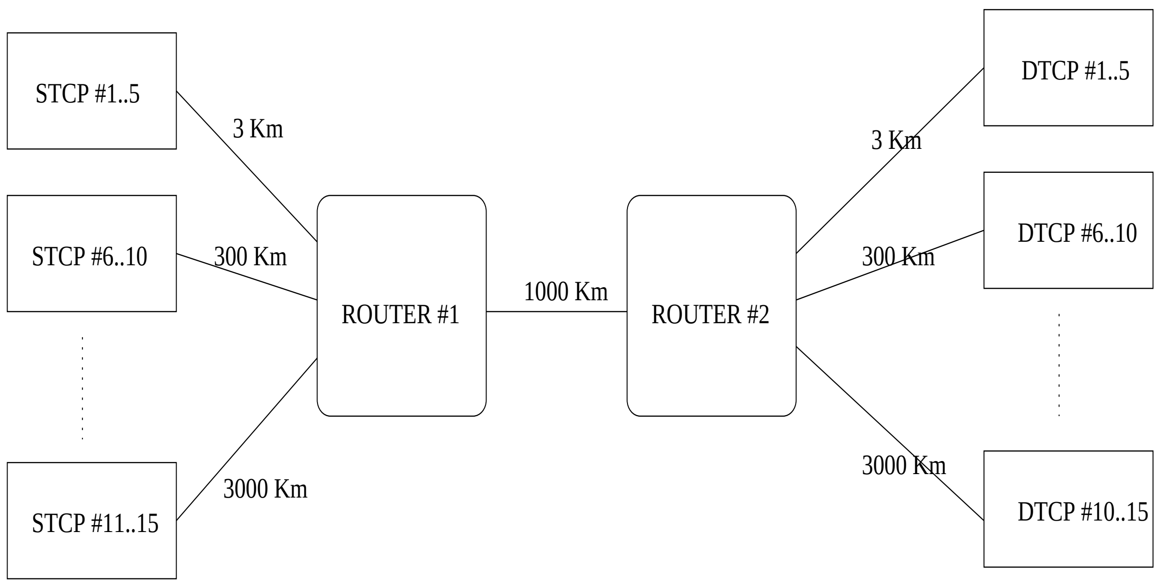
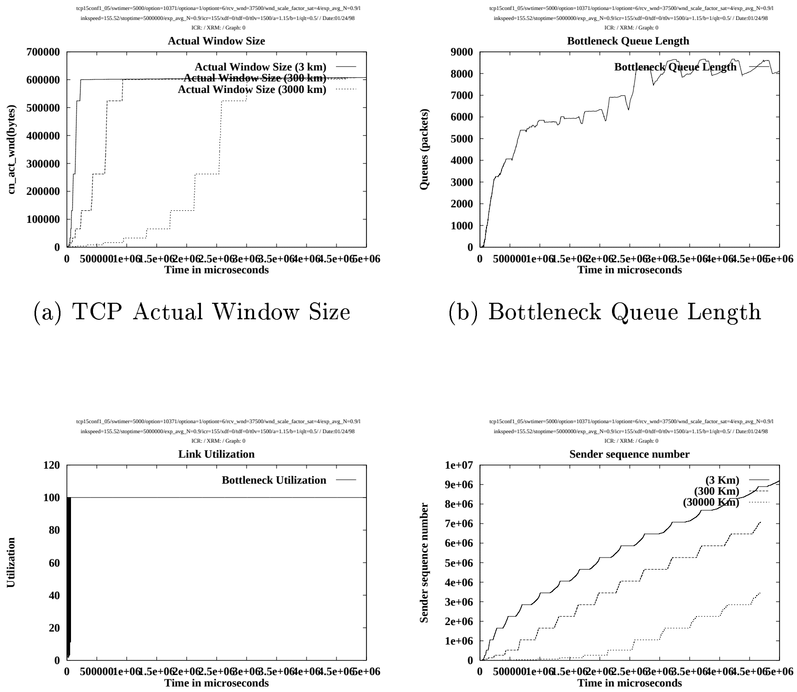


Figure 5: The NxN Heterogeneous RTT Configuration

We conduct five experiments with this configuration:

- In the first experiment (Figure 6) routers do not use explicit feedback (also called vanilla TCP).
- In the second experiment (Figure 7), the routers use the fixed-RTT translation method



(a) TCP Actual Window Size

(b) Bottleneck Queue Length

(c) Bottleneck Link Utilization

(d) TCP Send Sequence Numbers

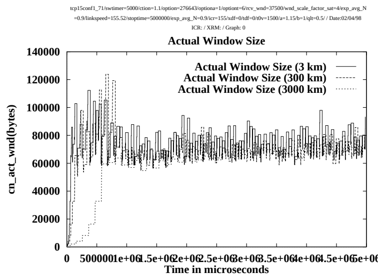
Figure 6: Simulation Results with NxN Heterogeneous Configuration without Explicit Feedback

to calculate the window feedback from the rate feedback for all sources. The fixed value chosen is 70 ms, which is the maximum RTT of any connection in this configuration.

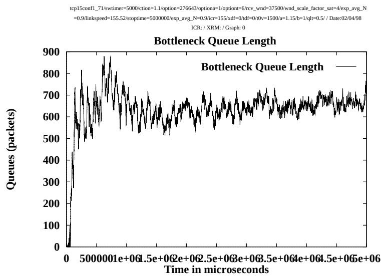
- In the third experiment (Figure 8) we use the translation using individual source RTTs. We assume that the routers have knowledge of the round trip delays, T_i for each of the sources and use that value to translate rate feedback to window feedback.
- In the fourth experiment (Figure 9) we use the ack-bucket scheme which simply clocks out the rate of acks based upon the rate calculated by ERICA+.
- In the fifth experiment (Figure 10) we combine the fixed-RTT translation with the ack-bucket scheme.

6.1.1 Performance of rate-to-window translation techniques

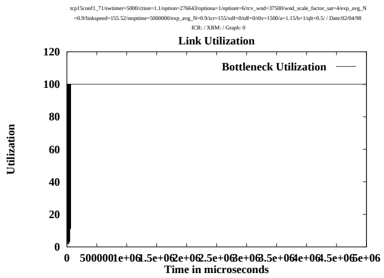
Figure 6(a) shows the TCP source's "actual window size" which is the minimum of the TCP congestion window (CWND), and the receiver window (Wr). Since the receiver window is constant at its initial value of 600000 (no TCP rate control), this graph is no different from a graph of CWND vs time. The window sizes increase exponentially (each increase spurt is twice the size of the previous one). However, between each increase spurt, there is a time gap which indicates that the source is waiting for new acknowledgments after sending the current spurt of traffic. This time gap increases exponentially as well because the first packet of the new spurt needs to travel though a larger queue to reach its destination.



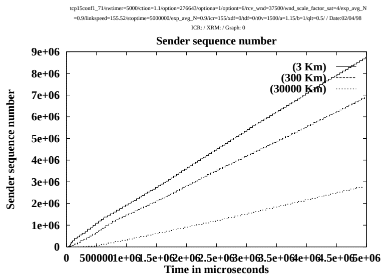
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



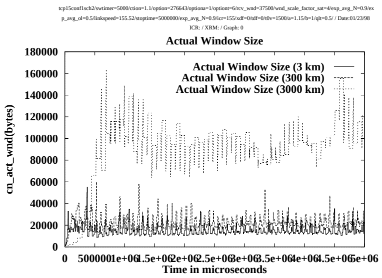
(d) TCP Send Sequence Numbers

Figure 7: Simulation Results with NxN Heterogeneous Configuration with Explicit Feedback using Fixed RTT=70 ms for Translation

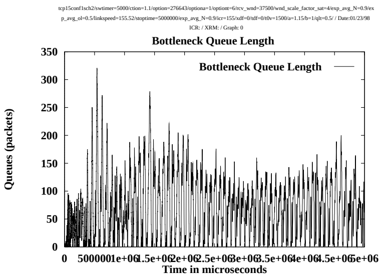
Figure 6(b) and 6(c) show the queue length, and link utilization respectively, while figure 6(d) shows sender sequence numbers of the various TCP sources. The round trip times measured at the source (not shown) increase (due to queue buildup in the network) and correspondingly slow the window increase by reducing the rate of acknowledgments returned to the sender. The queue lengths reach a maximum of about 8000 packets and round trip times measured at the source (not shown) went up to 600 ms. These quantities stabilized at this point as the sources reached the SSTHRESH (and receiver window) value of 600000 bytes and the congestion window increases no further.

Figure 7 shows a simulation of the same configuration where explicit feedback control is used. Notably, figure 7(b) shows queue lengths which oscillate between about 600 and 750 packets (an earlier experiment with fixed RTT = 30 ms (not shown here) exhibited even lower queues). The actual window size shown in figure 7(a) is now limited by the receiver window value (feedback) and not the congestion window value. Observe that the actual window size oscillates between 60000 and 80000 bytes (with fixed RTT = 30 ms, the values were 30000 and 40000 bytes).

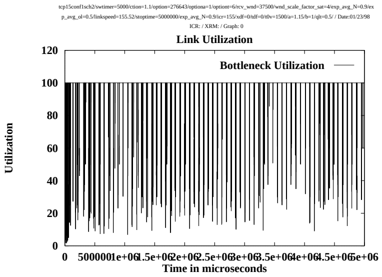
The queue length oscillation and the actual window size oscillation are related due to two reasons: a) ERICA+ is sensitive to queueing delays above the T0 parameter (which is 1.5 ms) [16], and b) the rate of acknowledgments which determines rate of change of window size and introduction of new packets is controlled by current queue levels. The queue oscillations indicate that the rate-to-window translation mechanism cannot fully control the rate of the TCP sources (other factors like the rate of acks also play a role). However the delay performance (and buffer



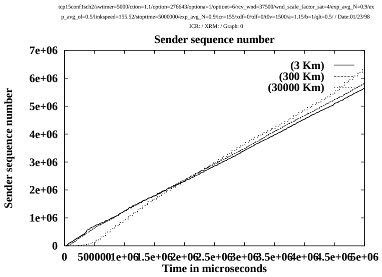
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



(d) TCP Send Sequence Numbers

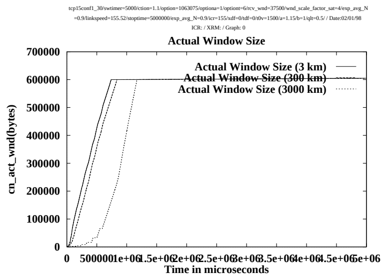
Figure 8: Simulation Results with NxN Heterogeneous Configuration and translation using individual source RTT

requirements) of the system display a marked improvement with rate feedback. Also note that the sequence number increase in figure 7(d) is much more smoother, which means that the burstiness of traffic is reduced.

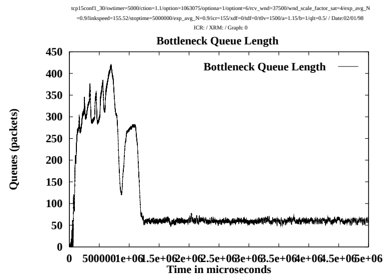
The graphs in Figure 7 show that the distribution of rate is unfair (see slopes of curves in figure 7(d)). We show in a later experiment that fairness can be achieved by adding ack-bucket control to this method. Note however, that if a smaller RTT were used for the translation (eg fixed RTT = 30 ms), even the addition of ack-bucket control does not achieve fairness because the long RTT sources cannot sustain the desired rate in an interval of length max RTT = 70 ms).

However note that this distribution, though unfair is an improvement over the uncontrolled TCP case where a connection can be almost completely starved. The overall utilization (figure 7(c)) is high because the smaller RTT connection grabs the available capacity. Queue sizes (figure 7(b)) are controlled due to the queue control feature of ERICA+ [3, 16] and the fact that window sizes are limited. The oscillations in actual window sizes (figure 7(a)) and queue lengths (figure 7(b)) which are interrelated by the rate of ACKs returning to the sources. We will examine combinations of ack bucket and single time value based translation in future papers, which can help smooth these oscillations and improve fairness.

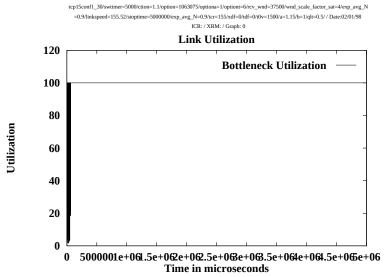
Figure 8 shows results with the translation scheme where the individual RTT (T_i) is used instead of a single value T. As expected the allocations are fair (see figure 8(d)), and the actual window sizes (figure 8(a)) are proportional to their RTTs. Queue sizes are also small (figure 8(b)), and utilization is not overly compromised (figure 8(c)).



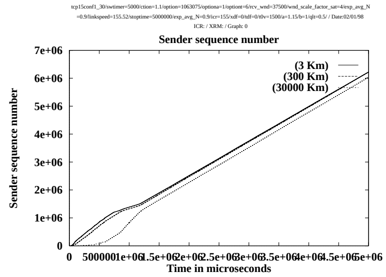
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



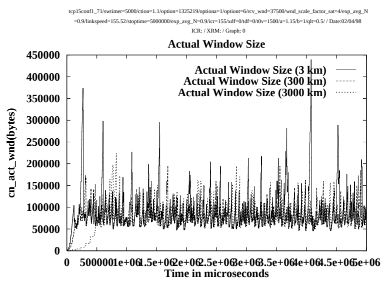
(d) TCP Send Sequence Numbers

Figure 9: Simulation Results with NxN Heterogeneous Configuration and Ack-Bucket Control

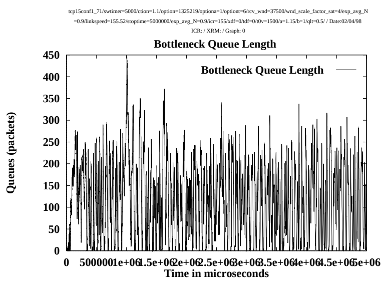
6.1.2 Performance of the Ack-bucket techniques

Figure 9 shows results where the ERICA+ algorithm calculates the explicit rates, but the acks are clocked out at the rate calculated by ERICA+, bypassing the rate-to-window translation. The performance is excellent, as expected from the discussion in section 4.2. The ERICA+ algorithm calculates the rate based upon not only the available capacity, but also the input rates and queue lengths. As discussed in section 4.2, this allows the ack-bucket scheme to be fair, efficient and to control queues. Specifically, the queue lengths in figure 9(b) are controlled to a low value of 50 packets, while the sequence number plot in figure 9(d) shows the fair distribution. Note however that the window sizes are not controlled i.e., the congestion window is the same as the actual window, and they rise to a maximum of 600000 bytes.

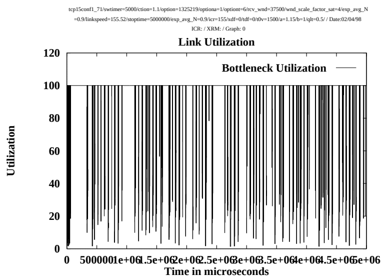
Observe that the congestion window size is the sum of the packets in the forward direction (controlled to small values by the ack-bucket method with ERICA+), and the acknowledgements in the reverse direction (which is large, but is stored efficiently at the intermediate router - only the ack numbers are stored in an array [4, 5]). Note that piggybacked acknowledgements cannot be stored (because the entire packet needs to be stored). In this case, the latest acknowledgement number is read from the packet into the array (the ack-bucket), and the last acknowledgement number sent to the source overwrites the acknowledgement number field. The checksum needs to be recalculated in this case as described in section 3. Note that a duplicate piggybacked acknowledgement is not considered by TCP as a duplicate acknowledgement and will not trigger the fast retransmit/fast recovery algorithms. Note this this method would share some of the problems of explicitly writing the window field, viz., cannot work with IPSEC or



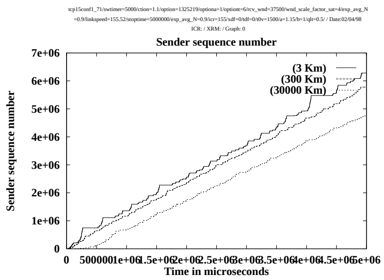
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



(d) TCP Send Sequence Numbers

Figure 10: Simulation Results with NxN Heterogeneous Configuration, translation using Fixed-RTT combined with Ack-Bucket Control

authentication, requires all acks to flow in the same path as the forward data flow.

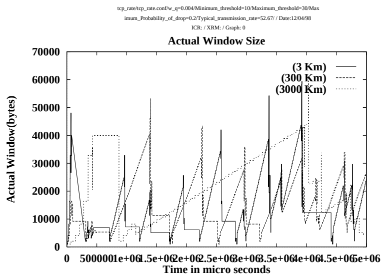
The problem of storing large number of acknowledgement numbers in the router can be controlled by combining the two schemes i.e, translation of rate-to-window using a fixed RTT, and controlling the acks using the acknowledgement bucket with ERICA+. Figure 10 shows the result of our next experiment which combines the fixed RTT translation scheme and the ack-bucket control with ERICA+.

Notably, the unfairness problem in the fixed-RTT translation-alone scheme is solved (figure 10(d)). Further, the drawback of ack-bucket, viz. the high congestion window values and long ack queues, is also taken care of (figure 10(a)). The queue lengths oscillate indicating burstiness (figure 10(b)), but are kept to low levels, while the average utilization is high despite some drops (figure 10(c)).

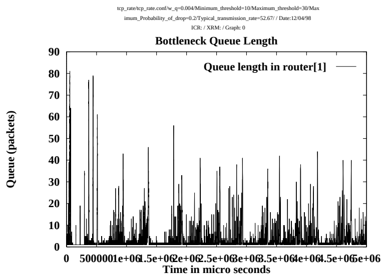
7 Random Early Detection

In this section we show a sample illustrative comparison with the same configuration as used in earlier simulations, and the queue managed with the Random Early Detection(RED) algorithm [13]. As mentioned earlier, a more detailed and careful comparative study with RED and ECN is presented in a recent work [23].

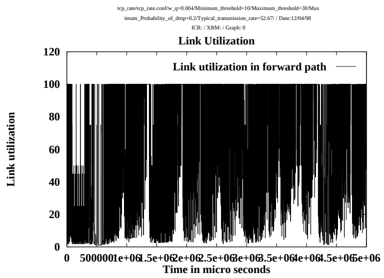
Our results are presented in Figure 11. As expected, RED keeps the bottleneck queue low.



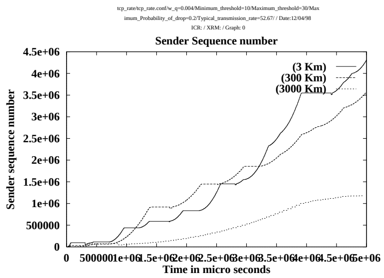
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



(d) TCP Send Sequence Numbers

Figure 11: Simulation Results with NxN Heterogeneous Configuration with Random Early Detection

Utilization, while relatively high is not as high as the rate-controlled cases. The difference in performance is clearly seen by observing the sequence number graphs, where the best rate-controlled schemes result in higher goodput. It is also observed that in terms of sequence numbers sent, the RED performance is not as fair as the rate controlled approach.

The RED parameters were chosen so that average queue length is between 10 and 30 packets. The exponential averaging factor was set to 0.004 based on the recommendation in the RED paper [13]. The maximum probability of drop was chosen to be 0.2 - this was also chosen keeping the recommendations in mind. Several other parameter choices yielded similar results. Since RED introduces packet drops early, whereas rate-control attempts to use a lossless scheme, the comparison may seem questionable. But we believe that the primary benifit of the rate-control approach is to avoid the complex dynamics caused by loss-based congestion control - which can be achieved given reasonable per-flow buffer sizing. With this in mind, the following simulation should be viewed as an illustrative comparison.

8 Discussion and Summary

We have proposed a new method of conveying explicit feedback to TCP sources. Our technique is to use an existing rate-based flow control algorithm ERICA+, effecting a rate-to-window translation, and writing the window feedback in TCP header of acknowledgments. This method does not require any changes to TCP header formats or the protocol, and can be implemented in both ATM edge devices/switches, as well as in enterprise routers. We have also considered a variant of a known method for TCP rate control, called the acknowledgement bucket which controls the rate of TCP connections by controlling the rate of the acknowledgements.

Our key contributions in this work are to demonstrate that rate-controlled TCP sources can achieve fairness and better delay characteristics in addition to high throughput performance. Auxiliary benifits include smaller buffer requirements and reduced burstiness of TCP traffic. We have studied the dynamics of rate-controlled TCP using both our method of explicit feedback and the method of ack-bucket control.

The TCP rate-control method does have its limitations. First, it typically requires per-flow state, and therefore is suitable only in situations where the number of flows are relatively small (eg a medium-sized corporate Intranet). Packeteer products can manage over 20,000 simultaneous flows - an evidence of reasonable scalability of the approach. Second, we require the capability to read and write into TCP packet and ack headers. This may not be feasible under future encryption schemes such as IPSEC, and an IP-header or MPLS-header based approach will be necessary. Diff-serv and MPLS provide such opportunities, and we are pursuing these directions [22]. Another related problem is that we require symmetry of paths, i.e., the TCP packets and acks should pass through the rate-control component. This restriction effectively eliminates the use of this approach in the interior routers of core networks. Inspite of these restrictions, TCP rate control appears to be an interesting supplement to the edges of enterprise network and there is a growing base of deployment of this approach. A detailed, comparative parallel simulation study is presented in [23].

Acknowledgements

This work was supported in part by DARPA contract number: F30602-97-C-0274, and by Packeteer, Inc.

References

[1] Shiv Kalyanaraman, Raj Jain, Sonia Fahmy, Rohit Goyal, "Performance and Buffering Requirements of Internet Protocols over ATM ABR and UBR Services," to appear *IEEE Communications Magazine*, June 1998.

[2] H. Li, K.Y. Siu, H.T. Tzeng, C.Ikeda and H. Suzuki, "A simulation study of TCP performance over ABR and UBR service in ATM networks," *Proceedings of INFOCOM'96.*

[3] 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.

[4] Paolo Narvaez and Kai-Yeung 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.

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

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

[7] Stevens, W. R., "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," January 1997.

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

[9] Ramakrishnan, K.K., Floyd, S., "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

[10] Floyd, S., Allman, M., and Partidge, C., "Increasing TCP's Initial Window," *IETF Internet Draft*, July 1997, Available as http://ds.internic.net/internet-drafts/draft-floyd-incr-init-win-00.txt

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

[12] Romanow, A., and S. Floyd, "Dynamics of TCP Traffic over ATM Networks," *IEEE Journal on Selected Areas in Communications*, Vol. 13, No. 4, May 1996.

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

[14] 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

[15] W.R. Stevens, "TCP/IP Illustrated, Volume 1," Addison-Wesley, 1994.

[16] R. Jain, S. Kalyanaraman, R. Goyal, S. Fahmy, and R. Viswanathan, "The ERICA Switch Algorithm for ABR Traffic Management in ATM Networks" *IEEE Transactions on Networking*, submitted, November 1997. Available from: http://www.cis.ohio-state.edu/ jain/papers/erica.htm

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

[18] Packeteer Inc., White papers on TCP rate control, http://www.packeteer.com/tcprate/

[19] Bobby Vandalore, Shivkumar Kalyanaraman, Raj Jain, Rohit Goyal, Sonia Fahmy, Seong-Cheol Kim, "Performance of Bursty World Wide Web (WWW) Sources over ABR," *(Poster in) WebNet '97*, Toronto, November 97, http://www.cis.ohio-state.edu/ jain/papers/webspec.htm

[20] Lixia Zhang, Scott Shenker, and D.D.Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *Proceedings of ACM SIGCOMM'91*, August 1991.

[21] IETF, "TCP Over Satellite (tcpsat) Working Group," http://www.ietf.org/html.charters/tcpsat-charter.html.

[22] S. Kalyanaraman, D. Harrison, S. Arora, K. Wanglee, G. Guarriello, "One-bit Feedback Enhanced Differentiated Services Architecture," *IETF Internet Draft, draft-shivkuma-ecn-diffserv-00.txt*, March 1998.

[23] P. Bagal, S. Kalyanaraman and B. Packer. "Comparative study of RED, ECN and TCP Rate Control," *Technical Report*, March 1999. Available from http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers-rpi.html