****

ATM Forum Document Number: ATM_Forum/98-0152R1

****

Title: Explicit rate control of TCP applications

****

Abstract

This contribution examines different methods of controlling the rate of TCP applications to achieve the goals of fairness, delay control and throughput. The key idea is to use an ATM ABR algorithm like ERICA+ 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 can be applied in ATM edge devices or in Internet routers. We explore 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.

****

Source:

Ramakrishna Satyavolu, Ketan Duvedi, Shivkumar Kalyanaraman
Department of ECSE,
Rensselaer Polytechnic Institute,
110, 8th Street, Troy, NY 12180-3590
Phone: 518-276-8979, Fax: 518-276-2433, Email: shivkuma@ecse.rpi.edu

The presentation of this contribution at the ATM Forum is sponsored by NASA Lewis Research Center.

The research was supported in part by DARPA contract number: F30602-97-C-0274.

****

Date: February 1998

****

```
Distribution: ATM Forum Technical Working Group Members (AF-TM)

**********************************************************************
Notice:
This contribution has been prepared to assist the ATM  Forum.  It  is
offered  to  the Forum as a basis for discussion and is not a binding
proposal on the part of any of the  contributing  organizations.  The
statements  are  subject  to change in form and content after further
study. Specifically, the contributors reserve the right  to  add  to,
amend     or     modify     the     statements     contained    herein.
**********************************************************************

A  postscript  version of this contribution including all figures and
tables has been uploaded to the ATM Forum ftp server in the  incoming
directory.   The postscript version is also available through our web
page:

      http://www.ecse.rpi.edu/Homepages/shivkuma
**********************************************************************
```

1 Introduction

There has been considerable interest in studying the dynamics of legacy TCP/IP applications over ATM (see [1, 2, 3, 4, 5] and references therein). These studies have indicated that an ABR service implementation can provide low-delay, fairness, and high throughput inside the ATM network for ftp-like applications, but the TCP/IP queues build up at the edge of the ATM network. In contrast, the UBR and 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.

The natural question which arises out of this is how one can carry the benefits of ABR end-to-end, i.e., provide better control of delay, throughput and fairness of TCP connections. An innovative approach suggested recently [4, 5] involves maintaining an "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 ACR of the VC carrying the TCP connection.

In this contribution, 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. As described later in this contribution, this mechanism can be used independently or in conjunction with the "acknowledgment bucket" mechanism. In this contribution we study the feasibility and benefits of this mechanism (in terms of throughput, delay and fairness), and its impact on TCP traffic dynamics.

We also identify a variant of the acknowledgement bucket idea whereby the acks are simply clocked out at the rate calculated by our ERICA+ algorithm. 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 end-system, and a simple queue-threshold based scheme to decide when to release the acknowledgements. The queue-threshold based scheme is needed 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 also study the combination of this scheme with the one of the TCP explicit feedback mechanisms.

In the public Internet where writing to TCP headers may not be a viable option, it may be possible to use rate-based techniques to control mechanisms scheduling, buffering and drop policies. These combinations will be studied in the future.

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)
- Little control over end-to-end delay characteristics
- 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)
- Traffic synchronization effects
- Burstiness of traffic depending upon reverse-path congestion
- Long ramp up times in long-delay bandwidth paths

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. 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 current interest in these metrics. Since the release of the ATM Traffic management specification 4.0 [14] it has been realized that some of the ideas from ABR flow control could be applied to improve the delay and fairness characteristics of TCP. Specifically, ABR switch algorithms like ERICA+ have demonstrated fairness and controlled delay characteristics using explicit rate feedback.

In this contribution, we study the feasibility of achieving similar goals in TCP (high throughput, fairness and low delay) by using rate control. The first technique we use involves calculating rate feedback just as in an ABR switch algorithm, but translating the rate into a window value to give feedback. This is a direct method of controlling the rate of the TCP source. We also consider another method of controlling TCP rate which is a variant of recent proposals [4, 5] called the "acknowledgment bucket" (ack-bucket) scheme. This method exploits the fact that TCP rate is controlled by the rate of receipt of its acknowledgments. More accurately, when the TCP's window size is fixed or slowly varying (eg: congestion avoidance phase), the rate of packets sent into the network is equal to the rate of receipt of acknowledgments. The ack bucket method can slow down but not prevent the increase of TCP window values. The TCP window controls the sum of unacknowledged segments in the forward path and the acknowledgements in the reverse path. We note that our explicit method of controlling TCP window can be used in conjunction with the ack bucket method. We study this issue further in section 4.1.1.

3 Procedure to Control 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. The rate r can be calculated using a switch algorithm like ERICA+ [16], or can be gotten by simply using ACR (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)

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 expect to build algorithms in the future with very little or no per-flow state. 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, note that there are some situations where the technique cannot be directly applied, eg, if the IP payload is encrypted (eg. using IPSEC), or authenticated. Also, since every acknowledgement needs to be marked, route flaps (due to instability in routing algorithms) can result in undesirable oscillations.

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 is Wr. In this state, 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. 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 describe 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, the allocation is fair and efficient. 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 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).

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. 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 rate of connection i, an unfair situation.

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

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. 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 to the ACR at the end system is insufficient, because it does not take into account the source queueing point.* 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 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. This 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. The key change we make to the algorithm is to limit rate increases to 10previous value. This step, which is similar to the limit imposed by the RIF parameter, is done to prevent burstiness which can result due to corresponding 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:

Initialization:

```
FairShare = 0;
MaxAllocPrevious = MaxAllocCurrent = FairShare;
```

End of Interval:

```
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
```

Note that other simplifications to ERICA+ are possible, especially for multicast configurations where it is undesirable to measure N [17].

6 Simulation Results

The simulations were run using the following parameters. For an explanation of ERICA+ parameter values, please refer [16]

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
Switch RTT estimate	30 ms
Link speed	155.52 Mbps
Maximum Segment Size (TCP: MSS)	1024 bytes
SSThresh (TCP)	600000 bytes

6.1 Simulation Results with a Heterogeneous RTT Configuration

We use the configuration shown in Figure 1 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).

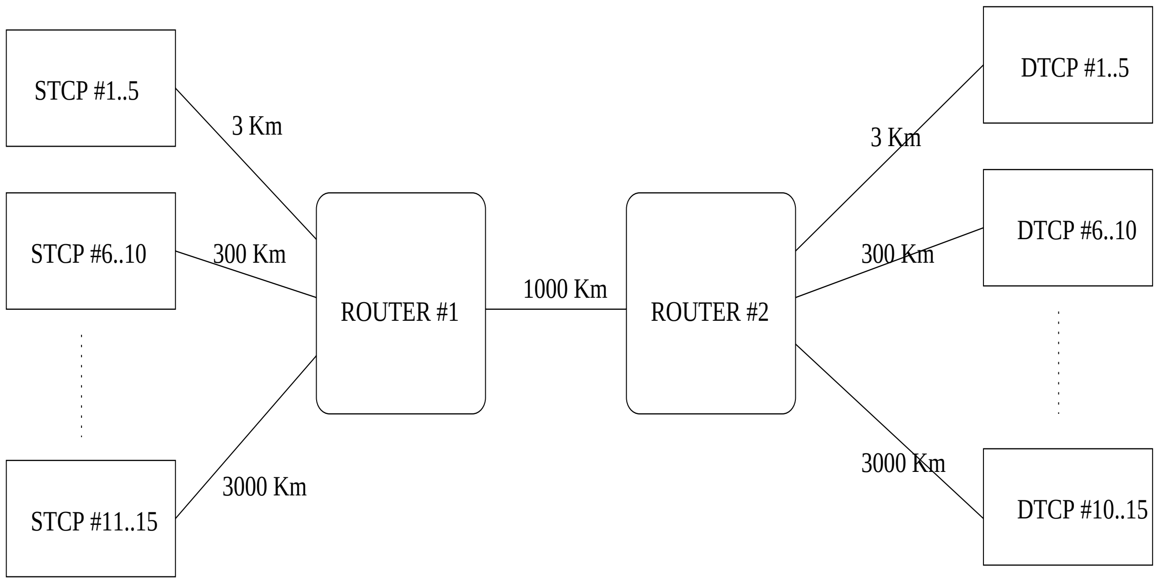


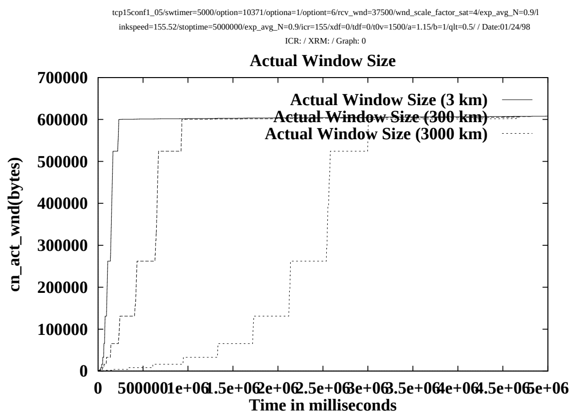
Figure 1: The NxN Heterogeneous RTT Configuration

We conduct five experiments with this configuration:

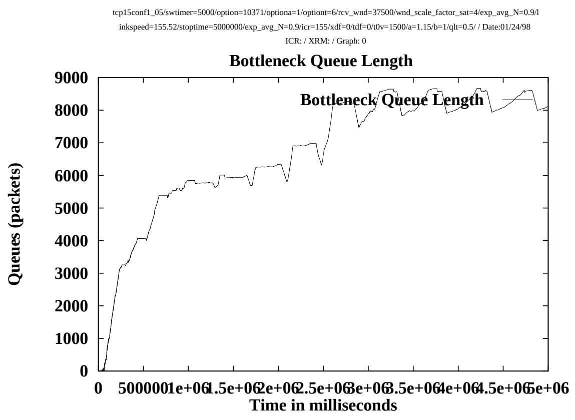
- In the first experiment (Figure 2) routers do not use explicit feedback (also called vanilla TCP).
- In the second experiment (Figure 3), the routers use the fixed-RTT translation method to calculate the window feedback from the rate feedback for all sources. The fixed value chosen is 70 ms, which is the maximum RTT of any connection in this configuration.
- In the third experiment (Figure 4) 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 5) 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 6) we combine the fixed-RTT translation with the ack-bucket scheme.

6.1.1 Performance of rate-to-window translation techniques

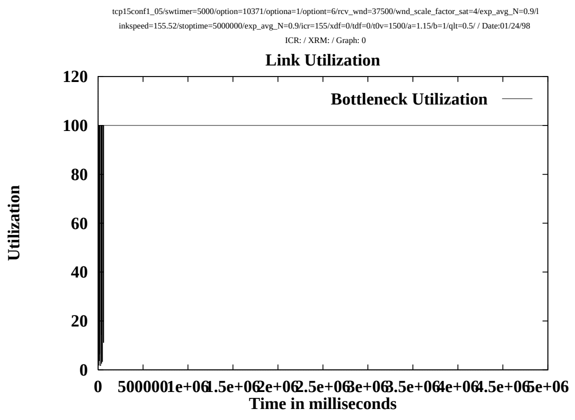
Figure 2(a) shows the TCP source's "actual window size" which is the minimum of the TCP congestion window (CWND), and the receiver window (Wr). Since the receiver window is constant at its initial value of 600000 (no TCP rate control), this graph is no different from a graph of CWND vs time. The window 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 the gap increases exponentially



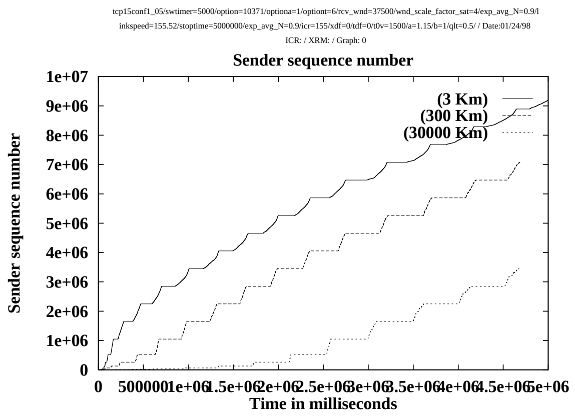
(a) TCP Actual Window Size



(b) Bottleneck Queue Length

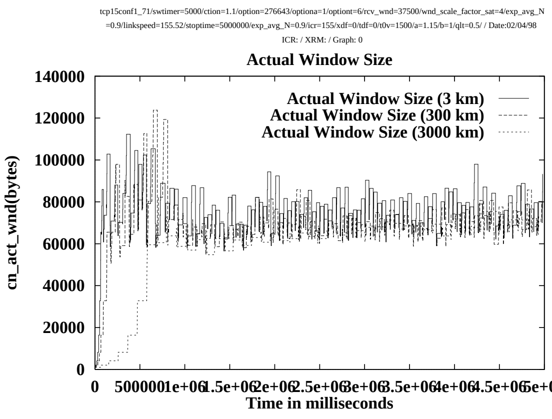


(c) Bottleneck Link Utilization

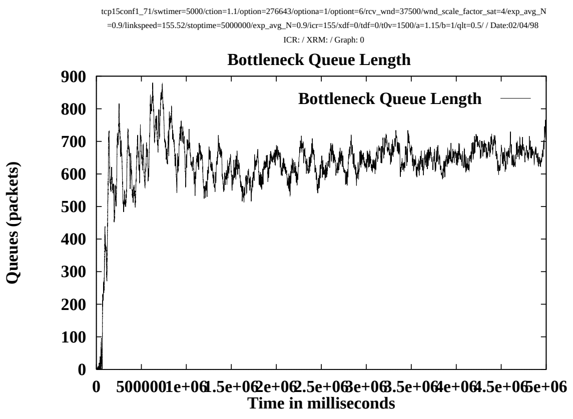


(d) TCP Send Sequence Numbers

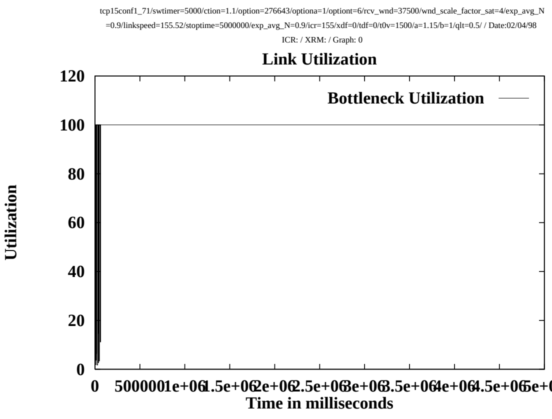
Figure 2: Simulation Results with NxN Heterogeneous Configuration without Explicit Feedback



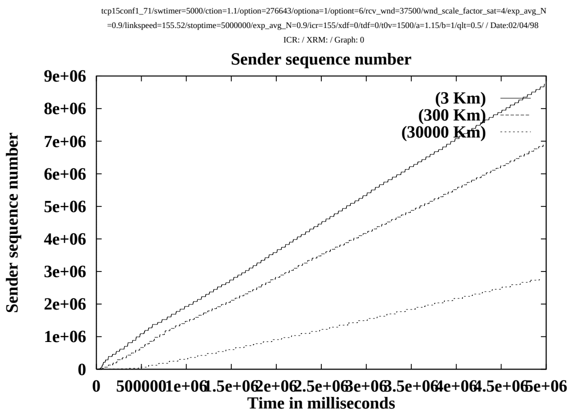
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



(d) TCP Send Sequence Numbers

Figure 3: Simulation Results with NxN Heterogeneous Configuration with Explicit Feedback using Fixed RTT = 70 ms for Translation

as well because the first packet of the new spurt needs to travel though a larger queue to reach its destination.

Figure 2(b) and 2(c) show the queue length, and link utilization respectively, while figure 2(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 600ms. 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 3 shows a simulation of the same configuration where explicit feedback control is used. Notably, figure 3(b) shows queue lengths which oscillate between about 600 and 750 packets (an earlier experiment with fixed RTT = 30 ms showed even lower queues). The actual window size shown in figure 3(a) is now limited by the receiver window value (feedback) and not the actual congestion window value. Observe that the actual window size oscillates between 60000 and 80000 bytes (with RTT = 30ms, 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 [16] (1.5 ms), 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 requirements) of the system display a marked improvement with rate feedback. Also note that the sequence number increase in figure 3(d) is much more smoother, which means that the burstiness of traffic is reduced.

The graphs in Figure 3 show that the distribution of rate is unfair (see slopes of curves in figure 3(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).

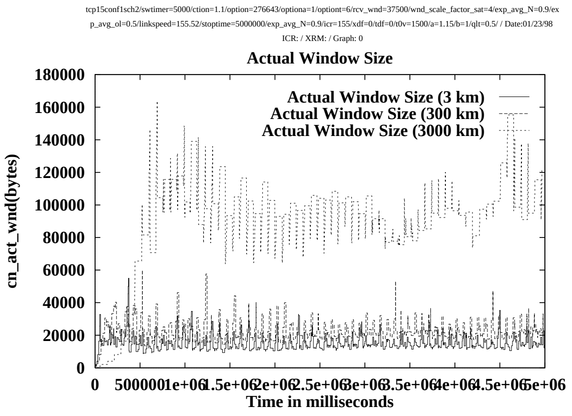
However note that this distribution, though unfair is an improvement over the TCP worse case where a connection can be almost completely starved. The overall utilization (figure 3(c)) is high because the smaller RTT connection grabs the available capacity. Queue sizes (figure 3(b)) are controlled due to the queue control feature of ERICA+ [3, 16] and the fact that window sizes are limited. The oscillations in actual window sizes (figure 3(a)) and queue lengths (figure 3(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 contributions, which can help smooth these oscillations and improve fairness.

Figure 4 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 4(d)), and the actual window sizes (figure 4(a)) are proportional to their RTTs. Queue sizes are also small (figure 4(b)), and utilization is not compromised much (figure 4(c)).

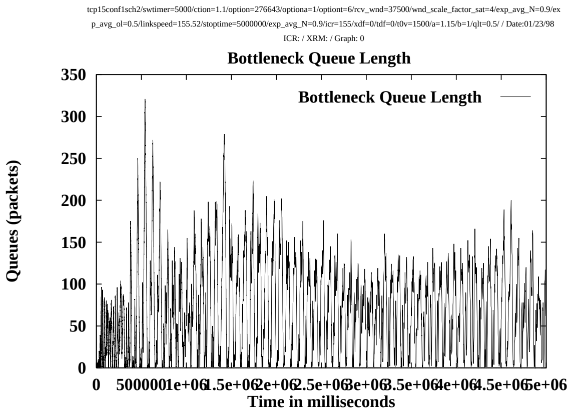
6.1.2 Performance of the Ack-bucket techniques

Figure 5 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 control queues. Specifically, the queue lengths in figure 5(b) are controlled to a low value of 50 packets, while the sequence number plot in figure 5(d) show 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.

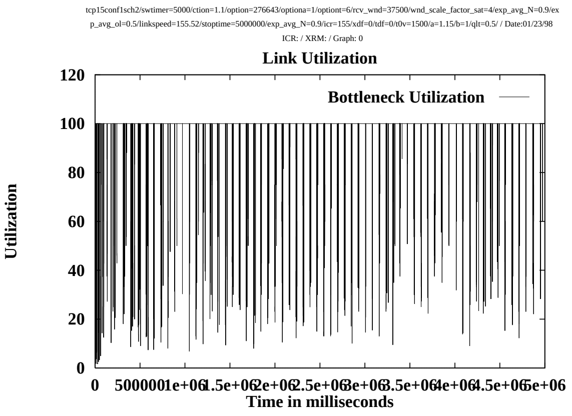
Observe that the congestion window size is the sum of the packets in the forward direction (controlled



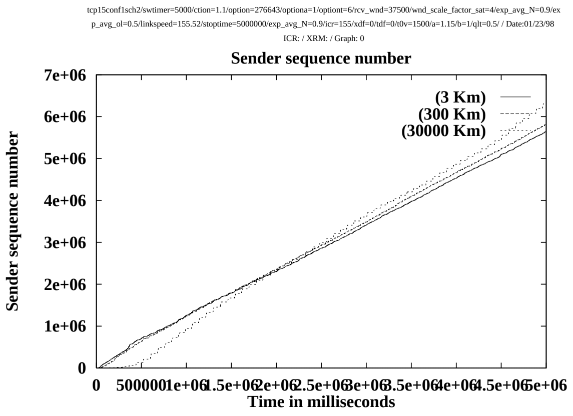
(a) TCP Actual Window Size



(b) Bottleneck Queue Length

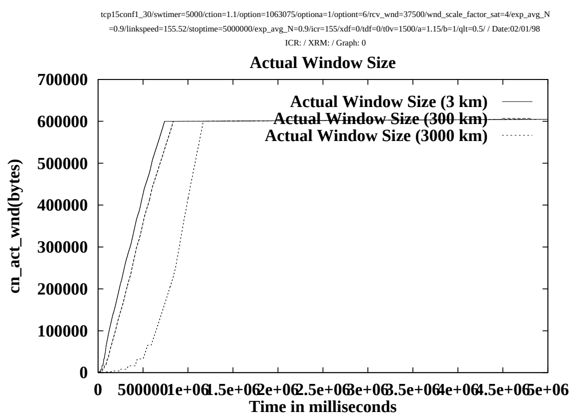


(c) Bottleneck Link Utilization

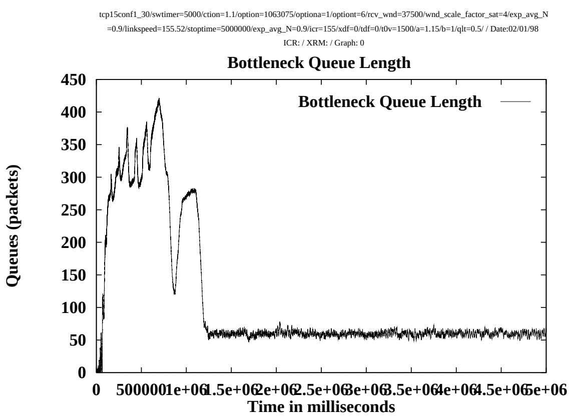


(d) TCP Send Sequence Numbers

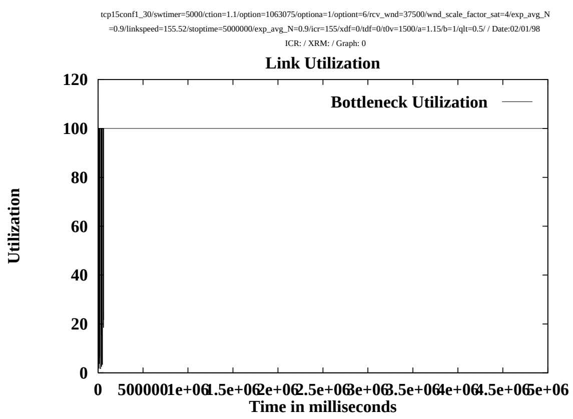
Figure 4: Simulation Results with NxN Heterogeneous Configuration and translation using individual source RTT



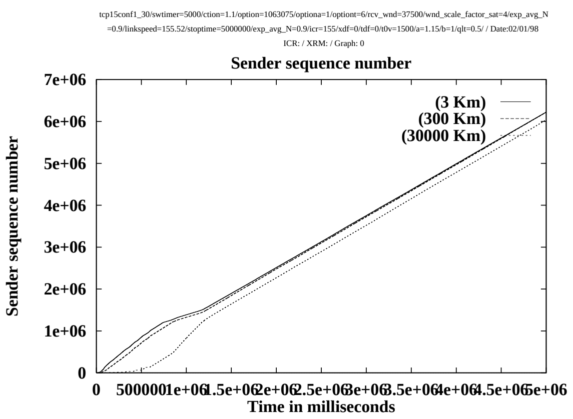
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



(d) TCP Send Sequence Numbers

Figure 5: Simulation Results with NxN Heterogeneous Configuration and Ack-Bucket Control

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 is 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 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 6 shows the result of our next experiment which combines the fixed RTT translation scheme and the ack-bucket control with ERICA+.

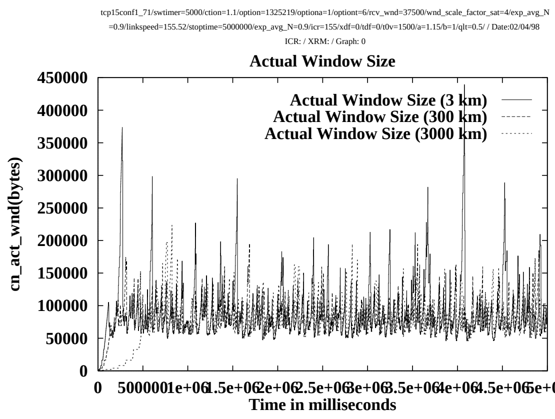
Notably, the unfairness problem in the fixed-RTT translation-alone scheme is solved (figure 6(d)). Further, the drawback of ack-bucket, viz. the high congestion window values, is also taken care of (figure 6(a)). The queue lengths oscillate (figure 6(b)), but are kept to low levels, while the average utilization is high despite some drops (figure 6(c)).

6.2 Simulation Results with a Generic Fairness Configuration-2

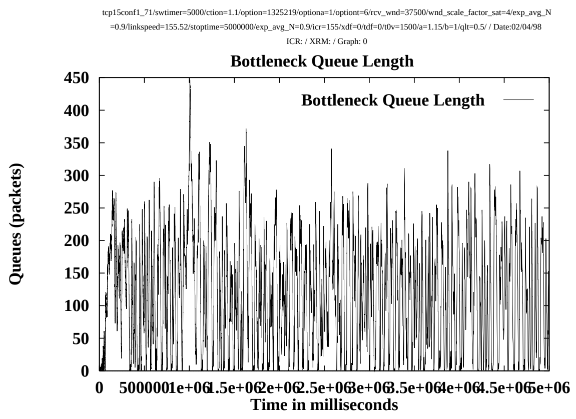
Simulations were performed on the GFC-2 configuration (Figure 7, see also [16]) to test the fairness of the different translation methods. The results are shown in figure 8. Figure 8(a) shows the distribution of sender sequence numbers for the case when no explicit feedback is used. While not much unfairness is seen, the queues (not shown) grow to large levels. Figure 8(b) shows the situation using a fixed-RTT translation (RTT = 130 ms, the largest RTT). The distribution is not fair, though the queue lengths are low (not shown). Utilizations are also somewhat lower in this case (not shown). However, the case where the translation is based on actual RTTs (figure 8(c)) shows a near-fair distribution. Queue lengths are low and utilizations are high too (not shown). Figure 8(d) shows the case for using the acknowledgement bucket, and figure 8(e) shows the combination of the fixed-RTT translation scheme and the ack-bucket. These achieve the fair rates too.

7 Summary and Conclusions

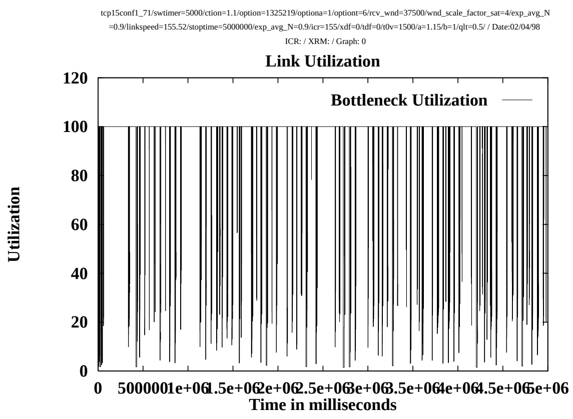
We have proposed a new method of conveying rate feedback to TCP sources, by translating it into a window value and writing the feedback in TCP header of acknowledgments. This method does not require any changes to TCP header formats or the protocol, and can be implemented in both ATM edge devices/switches, as well as in Internet routers. We have also considered a variant of a known method for rate control, called the acknowledgement bucket which controls the rate of TCP connections by controlling the rate of the acknowledgements.



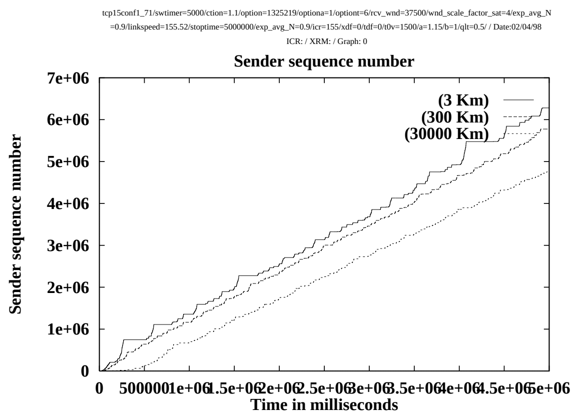
(a) TCP Actual Window Size



(b) Bottleneck Queue Length



(c) Bottleneck Link Utilization



(d) TCP Send Sequence Numbers

Figure 6: Simulation Results with NxN Heterogeneous Configuration, translation using Fixed-RTT combined with Ack-Bucket Control

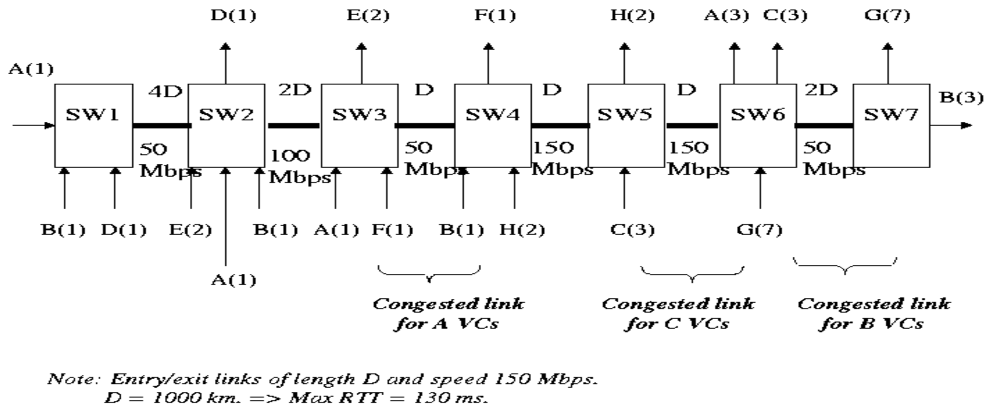


Figure 7: The Generic Fairness Configuration-2

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. Further, we have studied the dynamics of rate-controlled TCP extensively using both our method of explicit feedback and the method of ack-bucket control. Future work in this direction will include simpler switch algorithms, combination of simple rate-to-window translation schemes with ack bucket control schemes, and studies with limited buffer & drop policies such as RED, EPD etc. Translating the rate into a single bit feedback through ECN will also be considered.

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*, March/April 1998.

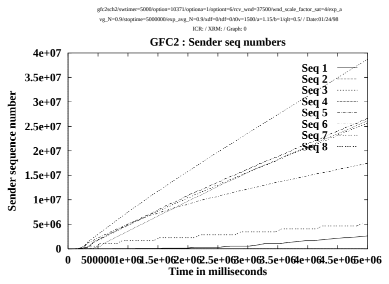
[2] H. Li, K.Y. Siu, H.T. Tzeng, C.Ikeda and H. Suzuki, "A simulation study of TCP performance over ABR and UBR service in ATM networks," *Proceedings of INFOCOM'96.*

[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.

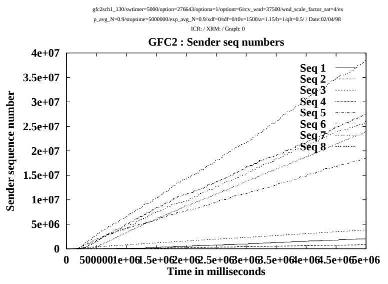
[4] Paolo Narvaez and Kai-Yeung Siu, "An Acknowledgment Bucket Scheme for Regulating TCP Flow over ATM," to appear in *Computer Networks and ISDN Systems Special issue on ATM Traffic Management*, 1998.

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

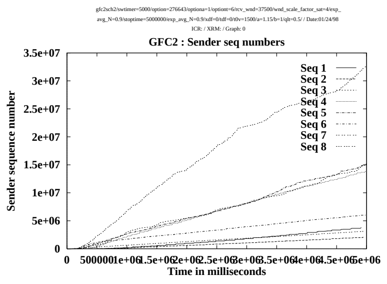
[6] V. Jacobson, "Congestion Avoidance and Control," *Proceedings of the SIGCOMM'88 Symposium*, pp. 314-32, August 1988.



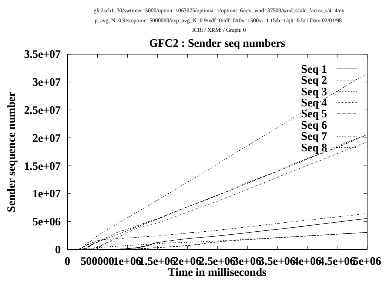
(a) No Explicit Feedback



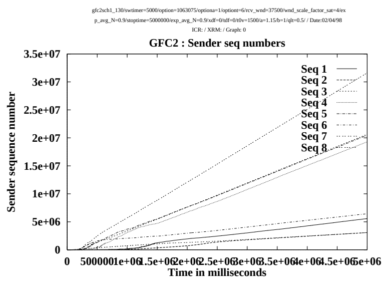
(b) Translation using fixed-RTT = 130 ms



(c) Translation using Individual source RTT



(d) Ack Bucket with ERICA+



(e) Combination of Fixed RTT translation and Ack Bucket with ERICA+

Figure 8: Simulation Results with GFC2 - Send Sequence Numbers

[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. [1]

[17] Sonia Fahmy, Raj Jain, Rohit Goyal, Bobby Vandalore, "A switch algorithm for ABR multipoint-to-point connections," *ATM Forum/97-1085*, December 1997,

[1] All our papers and ATM Forum contributions are available through http://www.cis.ohio-state.edu/˜ jain