ADAPTIVE RATE-BASED CONGESTION CONTROL VERSUS TCP-SS A PERFORMANCE COMPARISON

Rong-Feng Chang (1), Lap Huynh, James Gray (2)

(1) EE Department
National Cheng Kung University
Tainan, Taiwan 70101, R.O.C.

(2) IBM Networking Systems
P.O Box 12195, Research Triangle Park, NC, 27709, U.S.A.

Abstract

Recent simulation studies of TCP congestion control Slow-Start clearly show problems of oscillatory behavior in packet delay, congestion windows and throughput. Furthermore, it also suffers from rapid fluctuations in queue length, periodic packet losses, and underutilization of the transmission line in a large pipeline size and/or two-way traffic environments. These studies also exhibit three fairness problems in TCP-SS: systematic discrimination of a particular connection in periodic traffic, a bias against a connection with a long round trip time and a bias against bursty traffic. Based on the principles of rate control and preventive congestion mechanism, we have developed a new Adaptive Rate-Based (ARB) congestion avoidance scheme that avoids these problems present in TCP-SS. In this paper, we will demonstrate through simulations that ARB outperforms TCP-SS in two important areas: network efficiency and fairness.

1: Introduction

Advances in telecommunication and computer technology have radically changed the networking environment. For example, fiber optic transmission lines have dramatically increased transmission rates and decreased error rates. VLSI techniques have also cut the cost of memory significantly. As a result, networking architecture has changed constantly in keeping pace with these technological developments. Contrary to some speculations that this trend will create abundant bandwidth and congestion control will no longer be necessary, it only makes the congestion control more critical. One reason is that bandwidth comes with a cost, and no user can predict the peak bandwidth requirement nor they are willing to pay to support it. Another reason is that high-speed links will have to co-exist with low-speed links, and such networks will be more susceptible to congestion [13]. Furthermore, future high-speed networks will be required to carry different traffic types that have different service requirements. As a result, congestion control is not only necessary but a conventional schemes will not be enough to satisfy all service requirements while maximizing network efficiency at the same time [13, 14]. To make the most efficient use of the network resources in this new environment, we have developed a new end-to-end Adaptive Rate-Based (ARB) congestion control algorithm that is based on the principles of rate control and preventive mechanism (see [7] for references to other work in the area of adaptive congestion control).

TCP is the transport protocol of Internet, and its main function is to provide reliable end-to-end service over the underlying unreliable IP network. The current TCP congestion control, known as ***Slow Start (SS)***, was developed after the Internet experienced a series of congestion collapses in October of 1986 [8]. The Slow Start algorithm employs a dynamic windowing scheme instead of the fixed window scheme used in the original TCP. The Slow Start algorithm is now a part of the Internet Standard and has been implemented in most existing TCP/IP products. Recent simulation studies clearly show the following problems in the dynamic behavior of TCP-SS due to the nature of its congestion control [4,5,6,10,12].

- Rapid fluctuations in queue length, packet delay and throughput,
- Low link utilization in a large pipe size and/or a two-way traffic,
- Systematic throughput discrimination against particular connections in periodic traffic,
- A bias against a connection with a long round trip time,
- A bias against bursty traffic.

The focus of this paper will be on performance comparison between the two schemes, namely ARB and TCP-SS. We will demonstrate through simulations that ARB outperforms TCP-SS in achieving better overall network performance by having the following properties:

- Stability of queue length, delay and throughput.
- Fairness among competing connections/users.

This paper is structured as follows: we first give a brief overview of the two congestion control mechanisms (TCP and ARB) in section 2, we then proceed to present the comparison study in section 3, and 4; and

0-8186-3670-X/93 $03.00

finally, we draw our conclusions of the comparisons in section 5.

2: Overview of TCP-SS and ARB

TCP - Slow Start (TCP-SS): The following is a very abbreviated description of the Slow Start scheme. For further details, see [8]. The idea behind of the Slow Start scheme is using packet loss as an indicator of network congestion and resetting the window to one to avoid overwhelming a bottleneck resource. Precisely, when a time-out or packet loss is detected, the threshold variable, *ssthresh*, is set to be half of the current congestion window size, *cwnd*. Then, the congestion window size is reset to one. When the sender receives one ACK the *cwnd* increases by one until the window size reaches the *ssthresh*. This period is called congestion recovery phase. After that, the *cwnd* increases by one per round trip time (i.e., the increase is 1/*cwnd* per ACK). This period is referred to as congestion avoidance phase. In general, the window size is increased exponentially during the congestion recovery phase [10], and then linearly during the congestion avoidance phase. We summarize the dynamic window scheme of the slow start as follows:
When a new acknowledgement is received, the sender does

```
            if (cwnd < ssthresh)
                cwnd = cwnd + 1
/* Recover period, cwnd is doubled per rtt */
            else
                cwnd = cwnd + 1/cwnd
/* Avoidance period, cwnd is up by 1 per rtt */
            endif
```

When a dropped packet or time-out is detected, the sender performs

```
            ssthresh = cwnd/2
            cwnd = 1
```

Note the three key characteristics of the Slow Start algorithm, that cause TCP-SS to suffer from the problems that will be discussed later in this paper: 1) Non-paced window scheme: packets are **immediately** sent into the network upon ACK arrivals; 2) ACK clocking: ACK arrivals provide a clocking function to the sender so that the sender paces out packets at a rate less than the transmission rate at the bottleneck [8]; 3) Packet-based ACK: generally, in TCP-SS, each packet triggers the receiver to send an ACK. The congestion parameters are evaluated upon an ACK arrival.

ARB - Adaptive Rate-Base Congestion Control: In the following, only a brief description of the ARB algorithm is given, see [7] for further details and analysis. The basic idea behind this scheme, as the name implies, is to regulate the input traffic in the face of changing network conditions. In other words, when the algorithm detects that the network is approaching congestion (i.e., increasing delay, decreasing throughput), it reduces the rate at which traffic enters the network until the pre-congestion indications go away. And when the network is sensed to have enough capacity to handle the offered load, the algorithm adapts by allowing more traffic to enter the network but without exceeding the rate that the receiving end-point can handle.
The ARB scheme employs a closed-loop, distributed control mechanism based on information periodically exchanged between two end-points. The information exchanged consists of the minimum of two rates: one is the rate at which the receiver accepts arriving data from the network, and the other is the rate at which the receiver delivers data to the end user. The former indicates the status of the network/path and is used by the sender to regulate the input traffic load to avoid overloading the network/path and therefore causing congestion. The latter is used by the sender to avoid overloading the receiver's receiving capacity. The sender, based on this information, regulates the input traffic load accordingly.

ARB Parameters: The following is a list of the parameters that will be used to describe the ARB algorithm:

R_s is the rate at which the sender is allowed to transmit data into the network.
R_a is the average actual rate at which the data is sent into the network by the sender in one measurement interval ($R_a \leq R_s$, R_a could be less than R_s if there are periods where a sender has nothing to send).
R_r is the minimum of the rate at which the receiver accepts data from the network, and the rate at which it can process data (also described earlier in previous section). This quantity is measured by the receiver and returned to the sender.
B_s is the number of bits that the sender is allowed to send continuously in one burst at R_s rate.
B_t is the burst time $B_t = \frac{B_s}{R_s}$
M is the length of the measurement interval. M is used in computing the actual rate, $R_a = \frac{N_M}{M}$, where N_M is the number of bits sent during the measurement interval of length M.
R_{inc} is the increment rate. The sending rate is additively increased by R_{inc}, when ARB detects the network is underloaded.
T_{out} is the time-out value that a sender waits for a reply from a receiver.

ARB Algorithm: Using a rate control mechanism, ARB can generate a smooth transmission pattern compared with the window scheme in TCP-SS. This smooth pattern minimizes oscillations and congestions in the network. The transmission pattern is controlled by the burst size, B_s, which specifies the maximum length that

can be transmitted at one time. Given a value of a sending rate, R_s, ARB calculates the burst time, $B_t = \frac{B_s}{R_s}$. For each B_t period, ARB transmits up to B_s bits into the network, and waits for the next period. Thus, the bursts are separated by a gap of $B_t - \frac{B_s}{R_{peak}}$, where R_{peak} is the transmission rate of the outbound link at the sender.

The feedback mechanism, and frequency of measurements: At the end of the n^{th} measurement interval of length M, a sender sends a request, along with its measurement interval, (note that this request can be sent as part of the normal data being transmitted, i.e., a field in the header) for the rate information from the receiver. Upon receipt of this request, the receiver computes the receiving rate $R_r(n)$, which is the total number of bits received from the network divided by the amount of time M_r computed from the last request $(n-1)^{th}$ until the current request n^{th} (i.e., the elapsed time). The receiver then returns the rate R_r to the sender on the return path. The following figure illustrates requests and responses; it also shows how the rates are measured:

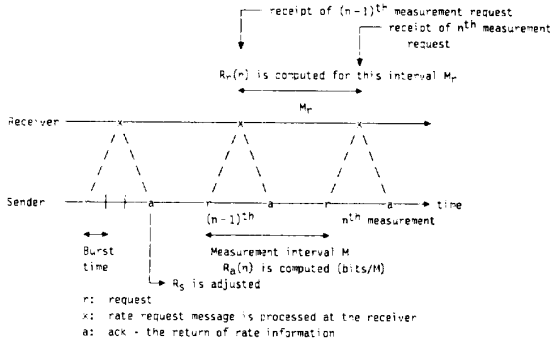

Figure 1. Rate Measurement

The measurement interval M determines the level of adaptivity to network conditions, the smaller M the better, but it is not without cost in terms of processing overhead at both the receiver and the sender of a connection. Therefore, setting M is a trade-off between adaptivity and processing overhead.

Adaptive mechanism-Adjustment of sending rate (R_s): The sending rate R_s is adjusted based on the feedback rate information sent back by the receiver, with an additive increase and multiplicative decrease algorithm.

We define three regions of operation that determine how the rate is adjusted based upon the feedback information, they are represented as Region 1 (R1), R2, and R3 respectively, as shown in figure below (Figure 2, the horizontal line represents the actual sending rate at a sender). The operating mode is set as follows:

- Set to R1 when $R_r \geq R_a - \Delta_R$. Δ_R is referred to as the sensitivity threshold ([7]).
- Set to R2 when $R_r < R_a - \Delta_R$.
- Set to R3 when a timeout waiting for a reply occurs (e.g., waiting time exceeds T_{out}). Note that the sending rate is cut by half when this situation occurs.

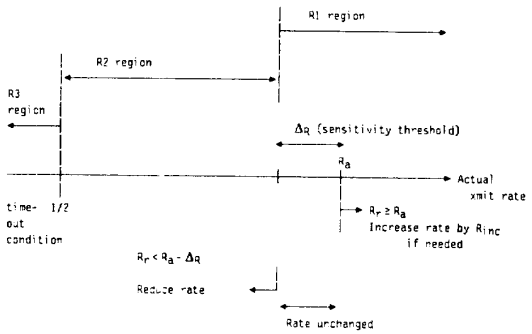

Figure 2. Rate adjustment based on feedback

Based on the feedback rate and the current operating mode, ARB adjusts the sending rate as follows:

$R_r \geq R_a$ and the current mode is R1: Additively increase the sending rate by R_{inc}, and the mode remains in R1.
$R_r \geq R_a$ and the current mode is not R1: no change, and the mode is set to R1.
$R_a - \Delta_R \leq R_r < R_a$: Leave the sending rate the same, and set the mode to R1.
$R_r < R_a - \Delta_R$: Multiplicatively reduce the sending rate to $\theta \times R_r$, and set the mode to R2.
Time out or packet loss is detected: Reduce the sending rate by half, and set the mode to R3.

One important feature of ARB is to protect a network from a phenomenon that is referred to as the ***queueing accumulation*** which can occur under the two following conditions: 1) Aggregated traffic at a bottleneck along the path gradually approaches to link capacity; 2) the difference between sending rate, R_a, and feedback rate, R_r, is less than Δ_R. Under this scenario, a sender does not change its sending rate and traffic at the bottleneck may start to saturate the link. If these two conditions stay for several subsequent measurement periods, the queue along the path would gradually build up. Moreover, if all the connections sharing the bottleneck do not reduce their sending rate for a considerable period, packets accumulated at the queue may eventually cause packet-loss.
We give a conceptual description about how ARB avoids ***queueing accumulation*** phenomena. The idea is that a receiver computes and monitors network delay

times, and accumulates changes in the delay time whenever it receives a rate request from a sender. If the accumulation of network delay is less than a certain threshold, a receiver computes a receiving rate and sends the rate to its sender. However, when an accumulation of network delay is greater than the threshold, the sender will be forced to reduce the sending rate and allow the queue at the bottleneck to decrease.

We summarize the ARB scheme as follows:
The sender algorithm consists of three components. One is to send data (if any) in each burst period, the second is to process responses received from the receiver, and the third is to handle time-outs caused by packet lost or congestion in the network or at the receiver. The receiver algorithm simply computes and returns the receiving rate.

Sender algorithm

Algorithm to send data

```
- If (more data can be sent within Bt) then
    If (M has elapsed and this is the first packet
        within a burst) then
      Build ARB segment
      -request receiver for rate information
      -Compute Ra sent during M
        /* bits sent divided by M */
    End
    Send data
  End
```

Algorithm to process rate acknowledgements from the receiver

```
- If (Rr < Ra - ΔR)
    Set rate Rs = θ × Rr
    Set operating region to R2,
      and reduce Rinc if necessary
  Else
    If (Rr ≥ Ra) and (operating region is R1)
      Increase rate Rs = max(Rs,Ra + Rinc)
      Rate should not exceed the maximum rate allowed
            Rs = min(Rs,Rmax)
    End
    Set operating region to R1
  End
- Compute new Bt
- Compute new M (if necessary)
```

Algorithm to handle time-out waiting for response, or packet loss

```
- Set operating region to R3, reduce Rinc to minimum
- Cut sending rate by half Rs = min(θ.5 × Rs,Rr)
- Compute new Bt
- Compute new M (if necessary)
```

Receiver algorithm

```
- Process the packet, add the size of this packet
    into the total number of bits received for the
    current measurement interval.
- If (rate information is requested)
    Compute Mr /* perform queueing accumulation
                  detection algorithm */
    Compute Rr /* number of bits received / Mr */
    Piggyback or send back a response with Rr
    Also respond if loss packets are detected.
  End
* Note that Mr is the measurement interval at the receiver.
```

3: Network Efficiency

In this section, our focus in the comparison of ARB and TCP-SS will be on network efficiency. The comparison is conducted on two traffic scenarios, namely the one-way traffic and the two-way traffic that were studied in [10,12]. The network configurations will be the same as the ones used in [10,12].

3.1: Network Configuration

To provide an unbiased comparison, both ARB and TCP-SS use the same configurations as in [10, 12]. Shown in Figure 3, the network configuration consists of two switching nodes (routers) connected by a full duplex low speed link with a bandwidth of 50 Kbps. Its propagation delay is either 0.01 second or 1 second representing two scenarios, a short pipeline and a long pipeline, respectively. The links between hosts and switching nodes have a bandwidth of 10Mbps and a propagation delay of 0.1 msec. We assume all links are error-free.

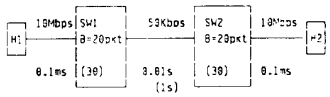

Figure 3. Network Configuration

The buffer size associated with each outgoing link in each switching node is 20 packets for all simulations, except the link that simulates 10 connections uses a buffer of 30 packets. We do not consider the buffer sharing between the different outgoing links. Each node performs a FIFO service discipline and uses a ***drop-tail discarding scheme***. i.e., discarding an arriving packet when the buffer is full.

We use a heavy traffic assumption for both TCP-SS and ARB. That is there are always packets waiting to be sent at the hosts. The data packet size and ACK packet size are fixed with 500 bytes and 50 bytes, respectively. For TCP, each connection has a *maxwnd* value of 1000. In our simulation configurations, the pipeline size is 0.125 and 12.5 packets for the propagation delays of 0.01 sec and 1 sec, respectively.

One-way traffic is defined as all the sources located on host-1. On the other hand, two-way traffic means that some data sources are in host-1, and some are in host-2. For the following simulations, TCP-SS does not use a delay-option. That is a receiver sends back an ACK immediately after receiving a data packet.

3.2: One-Way Traffic

TCP-SS: The following simulation results are reported verbatim from [10]. Most discussions regarding

TCP-SS are also summarized from the same paper. Due to the limited space, we only show one case, which is three TCP connections, all sources sit in host-1 and their destinations located on host-2. The propagation delay is 1 sec, and the switches have a buffer size of 20 packets. Note that the same phenomena persist in other configurations. Detailed information is given in [10].

As shown in figure 6, two unexpected phenomena are found: 1) clustering of packets from each connection, 2) synchronization of packet losses.

From figure 6-A, one can observe both periodic packet losses and synchronization of packet losses, where the symbols of cross, star and box represent the time of connection 1, 2 and 3 losing a packet, respectively. The periodic packet losses result from the nature of the TCP-SS congestion control algorithm. This scheme tends to overutilize the network capacity by increasing the window size until a discarded packet is detected. Then, the window is reset to one and the cycle begins again (shown in figure 6-B).

The corresponding link utilization is 90% [10]. In fact, in one-way traffic, the link utilization is a function of pipeline size. When the window size is less than two times the pipeline size, no packets queue in the buffer. That is the server is idle. Thus, the idle time increases and the link utilization decreases as the pipeline size increases.
Increasing the buffer size can increase the link utilization level. However, in later section, it will be shown that this is not true in two-way traffic case. Note that although high link utilization can be achieved for one-way traffic by a larger buffer, the amplitude of oscillation also increases, and the result will be an increase in the variance of throughput and packet delay.

The second important characteristic of TCP dynamic behavior is the clustering of packets from a single connection. In other words, each TCP connection sends out a full window's worth of packets in one burst and then waits for the next cycle. This clustering effect is reflected in the congestion window trace in figure 6B. Looking closely, one can find that the window traces alternatively stay in a constant region, and only one connection increases at a time. Since the congestion window only changes upon the receipt of ACK's, this indicates that all of a connection's ACK's arrive in a cluster. Refer to [10] for an analysis of this effect. The impact of the clustering effect in rapid queue fluctuations and in fairness will be shown in section 3.3, and section 4 respectively.

ARB: Running the same configuration, we show the ARB simulation result in figure 7. The increment rate, R_{inc}, is equal to 900 bps, the reduction factor $\theta = 0.95$, and the sensitivity threshold Δ_R is set to a rate that allows two packets to queue in each node along the path. In addition, the burst size is equal to one packet size. The upper chart in this figure is the trace of queue length at the gateway, and the lower chart is the trace of the sending rate for connection 1.

Compared with the results of TCP-SS in figure 6 with ARB in figure 7: first, no packets are lost in ARB due to the ARB congestion prevention mechanism; second, as shown in this figure, ARB greatly reduces oscillatory behavior in the queue length, the sending rate and the packet delay time. In fact, the maximum queue length is 11 packets (not shown in the figure), incurred at the initial transit period. The mean queue length has a very small value of 2.72 packets, and its variance is 1.6. Although figure 7 shows a drift downward in rate, the trace of rate also drifts upward around 350 second. It is not shown in this figure since we only show a part of trace. In fact, the rates of the three connections oscillate around 16000 bps each. In TCP-SS, the maximum queue length is equal to the maximum buffer size, and its variance is much larger than ARB. From the queue length comparison, we can see a significant difference between a congestion prevention mechanism, which operates the network around the Knee, and a congestion reaction mechanism, which pushes the network over the cliff into the congestion region and recovers it later.

As shown, with ARB the link utilization is 0.98 while the queue length is kept in a very small region. Note that, the link utilization of TCP-SS is 0.9, which includes the retransmission traffic due to packet losses. Therefore, even though TCP and ARB have the same link utilization, ARB still outperforms TCP-SS in useful throughput.

Note that the clustering effects in TCP-SS are eliminated in ARB. This is because the packets are paced out by a local timer by the sending rate. The packets from ARB are not allowed to enter the network in a back-to-back manner when the burst size is equal to packet size. Although, in the case that burst size is greater than one packet size, more than one packet can be sent in a cluster, the size of cluster is limited. In fact, the maximum size of a cluster is equal to the burst size. In contrast, the cluster size of TCP-SS is equal to the full window size. This cluster effect plays a key role in causing rapid fluctuations in queue length in two-way traffic for TCP-SS.

3.3: Two-Way Traffic

In this section, we present the results of two-way traffic. Comparing with one-way traffic, we observe two more unexpected phenomena in two-way traffic [12]. There are rapid fluctuations in the queue length and significant underutilization of the link. For example, the link utilization of TCP-SS decreases from 90% in one-way traffic to 60% in two-way traffic. In contrast,

ARB gives outstanding performance in two-way traffic. Compared with 60% utilization in TCP-SS, ARB can achieve a remarkable link utilization, 98%.

TCP-SS: As described in section 3.2, two notable facts in TCP-SS, clustering and loss-synchronization, persist in two-way traffic. In addition, we find two new phenomena in two-way traffic: 1) ACK-compression, 2) queue-synchronization. ACK-compression results from the interaction of data and ACK packets and causes a rapid fluctuation in queue length; queue-synchronization keeps the link in a significant underutilization level even if we increase the buffer size. Unlike one-way traffic, a large buffer does not increase link utilization. In this report, we only discuss briefly the impact of ACK-compression, refer to [12] for a more detail discussion on these two phenomena.

We directly import figure 8 and figure 9 from [12]. Figure 8 depicts two traces of queue lengths with a propagation delay value of 1.0 sec, representing pipeline size = 12.5 packets. The upper trace indicates the forward (from host-1 to host-2) queue length; the lower trace represents the reverse queue length. Two connections are considered: One has its source on host-1 and the other has its source on host-2. The buffer size at switches is 20 packets.

Comparing the performance of one-way traffic in figure 6 with figure 8, we find two dramatic differences, a rapid fluctuation in the queue length and a significant idle time on the transmission line. These features imply:

High variance in both queue length and end-to-end delay.
Rapid fluctuation in the throughput.
Underutilization on the bottleneck link.

Similarly, the same phenomena are exhibited in figure 9. This figure shows the queue lengths at the switches 1 and 2. The configuration has ten connections. Five connections have their source on host-1, and the other five connections have their source on host-2. The buffer size for each outgoing link is 30 packets. The propagation delay is 0.01 sec, corresponding to a pipeline size of 0.125 packet.

ACK-compression: In the one-way traffic case, ACK arrivals can provide a reliable clocking function because it is the only traffic on the return path. But in the two-way traffic scenario, the ACKs do encounter nonempty queues because of the traffic in the other direction. As a result, the clocking function is broken, and the arrival rate of ACKs cannot reflect the service rate at the bottleneck. The spacing time between two adjacent ACK's are disturbed by the data packets.

Looking closely, since ACK packets are smaller than data packets (10 times smaller in this simulation), the arrival interval time of ACKs at the source will be much shorter than the spacing time of data packets at the queue. The arrival of each ACK immediately triggers transmission of 2 data packets (i.e., during the congestion recovery period). Thus, a cluster of data packets will be generated, and their spacing time is roughly equal to the ACK's spacing time at the queue, which is 1/10 of data transmission time in this simulation. The sharp increases in figure 8 and figure 9 indicate these ACKs closely cluster around data packets hitting the queue [12]. This phenomena is referred to as the ACK-compression.

In short, the rapid queue fluctuations in two-way traffic, square-wave-like curve fluctuations, result from ACK-compression. In the previous discussion of ACK-compression, we only use two assumptions: 1) The size of an ACK is significantly less than the size of a data packet, 2) The packets from a connection are clustered together. The first one exists in almost all network protocols. The second one is valid in congestion control using a nonpaced window scheme. Therefore, in [12], the authors pointed out that rapid queue fluctuations can be expected over a wide range of nonpaced window congestion control schemes in two-way traffic.

Another interesting feature in two-way traffic is that significant idle time on the link is shown in figure 8, resulting in underutilized links: for small pipeline size, a propagation delay of 0.01 second, the utilization of the bottleneck line is 70%, compared with nearly 100% in one-way traffic. (not shown in this report); in figure 8, for large pipeline size, a propagation delay of 1 second, the utilization of the link reduces to 60%, compared with 90% in one-way traffic.

Note also that increasing the buffer size will not improve the link utilization in two-way traffic. This is because the idle cycle is a function of the **effective** pipeline size. When we increase buffer size, the effective pipeline size also increases due to the increase of the ACK delay. Therefore, a large buffer does not reduce the proportional idle period [12]. For example, in one of the simulation results, when the buffer size is increased from 20 to 60 and 120, the utilization remains at 70%.

ARB: Two-way simulation results of ARB are shown in figure 10 for a large pipeline of 12.5 packets, and figure 11 for a small pipeline of 0.125 packet. The figures show traces of forward and reverse queue length in the gateway. Again we use the same configurations as TCP-SS. Two connections are considered in figure 10. The congestion control parameters are R_{inc}=2.5Kbps, and the burst size = the data packet size.

Compared with the TCP-SS performance in figure 8, the result of ARB shows a dramatic improvement. First, no packet losses are found. Second, the amplitude of oscillation in queue length is reduced roughly from 20 packets in TCP-SS to 2 packets in ARB, as shown in figure 10. The maximum queue length is 7 packets

versus 20 packets in TCP-SS. In addition, ARB has an average queue length of 1.95 packets and a standard deviation of 0.93. Thus, ARB greatly outperforms TCP-SS in packet delay time and throughput. Third, the average link utilization achieves a level of 0.99, compared with 0.6 in TCP-SS.

Figure 10 shows a drift upward in queue length. This does not indicate that the queue length steadily increases. In fact, when the queue length is greater than 3 packets, senders detect this pre-congestion signal and cut back their rate. Thus the queue length is reduced to zero and the cycle start again. The cycle is not clear in the figure because a small part of the trace is shown.

In figure 11, ten connections are carried over a small pipeline. Each connection uses the same congestion parameters which are R_{inc} = 500 Bps, the burst size = the data packet size.
Compared with the TCP-SS results in figure 9, the results of ARB reveal improvements similar to those shown in the large pipeline. For example, the amplitude of oscillation in queue length is reduced roughly from 30 packets in TCP-SS to 5 packets in ARB, as shown in figure 11.

Unlike TCP-SS, the performance of ARB is not a function of pipe size, as mentioned previously. Therefore, the impact of pipeline size on performance is minimized.

Summary: TCP-SS suffers from the following problems:

Periodic packet losses.
Oscillating behavior in queue length, congestion window size and packet delay time.
Underutilization in two-way traffic and/or a large pipe size.

These problems are due to the nature of TCP-SS:

Uses a congestion reaction mechanism.
Resets the window to one when a packet loss is detected.
Uses a nonpaced window scheme.
ACK arrivals are assumed to provide a reliable clocking function.

ABR offers a significant improvement in these areas by:

Use of a congestion prevention mechanism.
Use of a rate control mechanism.
Use of explicit feedback rate to adapt to network traffic.

4: Fairness

Recent simulation studies [4,5,6] reveal three fairness problems in TCP-SS: 1) a systematic discrimination against some connections in periodic traffic, 2) a bias against connections with longer round trip times, 3) a bias against bursty traffic. Basically, the first problem is caused by the nonpaced window algorithm in TCP-SS. The second problem reflects the nature of the window adjustment scheme in TCP-SS. The clustering effect mentioned in section 3 plays a crucial role in the third problem.

As expected, ARB can greatly relieve all three problems because it uses a rate control mechanism to pace out the packets and employs an effective rate adjustment scheme. This adjustment scheme adopts an additive increase and multiplicative decrease algorithm, which has been shown to achieve both, efficiency and fairness goals [3].

The comparison will be based on the same network configuration as the one used in [4,5]. Only brief explanation for the fairness problems in TCP will be given, for a more detailed analysis, we encourage the reader to consult [4,5].

4.1: Network Configuration

We consider the following simple configuration depicted in Figure 5. The network consists of three hosts and one gateway (router). The traffic from two high speed lines, 8000 Kbps, are feeding into one slower link, 800 Kbps. Propagation delay is a crucial parameter in the fairness simulations. d_{13}=5 msec and d_{34}=100 msec represent the propagation delay between node 1 to node 3 and between node 3 to node 4, respectively. The propagation delay between node 2 and node 3, d_{23} is an adjustable variable.

Each host has a FTP (File Transfer Program) source, which represents a bulk data transfer, i.e., the data size is larger than the pipe size. In the simulation, we assume FTP sources always have packets to be sent. The packet size of FTP data and ACK is 1K bytes and 40 bytes, respectively. To transmit a FTP data packet, the gateway takes 10 msec. The buffer size in Gateway-3 is 15 packets. In this configuration, without a queueing delay, the round trip time for source 1 to sink 4 is equal to 221.44 msec.

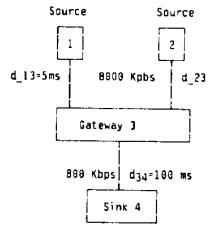

Figure 4. Simulation Network Configuration.

Two gateway congestion control schemes are considered in TCP-SS: ***Drop Tail*** and ***Random Drop***. The former one was mentioned in section 3. The random drop scheme was originally proposed by Jacobson and

is intended to improve fairness [9]. The random drop scheme may improve fairness in periodic traffic scenario, but it cann't solve the biases due to a long RTT and bursty traffic as shown in the following.

4.2: Bias in Periodic Traffic

TCP-SS: In periodic traffic, we investigate a steady-state setting, where the maximum window size for each FTP source is set to be 32 packets. The simulation result is shown in figure 12, where the x-axis and y-axis represent the ratio between node 2's and node 1's RTT and the percentage of node 1's average throughput of the maximum possible throughput, respectively. Each dot in the figures indicates a result of 100 seconds for various RTT ratio. Node 1's RTT is fixed, and node 2's RTT is varied.
As shown in figure 12, highly biased throughput occurs. Node 1 gets about 90% of the bandwidth for most RTT ratios. However, some RTT ratios cause node 1's bandwidth to drop to only 10-20%. The valleys occur periodically. Interestingly, the period is 10 msec, equal to the transmission time of FTP data packets at the gateway. In other words, the figure reveals that, in TCP, a slight change of RTT can cause a drastic change in throughput. In [4], a clear explanation of this phenomena was given.

We summarize the explanation as follows: Basically, this bias results from the nonpaced window scheme in TCP-SS. In a drop tail gateway, a new arrival will be dropped when the queue is full. Precisely, the dropped packets are dependent upon the timing of arrivals at the full queue. In the above configuration, the arrival time of a packet is determined by the departure time of previous packets due to a fixed transmission time (fixed data packet size), propagation delay and a nonpaced window scheme. Therefore, given a particular propagation delay, an arrival pattern can be built so that the dropped packets are always from one connection instead of being uniformly distributed over two connections. For example, in figure 12, when the round trip ratio ranges from 1.039 to 1.045 (the RTT of node 2 from 230 ms to 2231.44 ms), the dropped packets are always from node 1. Obviously, this connection will receive a very poor throughput since most of the time its window size is very small. Its window is reset to 1 when a dropped packet is detected. Note that a slight change of propagation delay time, in the order of the data transmission time at the bottleneck, can completely change the pattern of arrival. Then the network completely changes the bias connection from node 1 to node 2. Therefore, a drop tail gateway is very sensitive to small changes in network parameters.

According to the above discussion, this bias can be fixed if a random drop gateway policy is used. With this policy, when a packet arrives at a full queue, a packet in the queue is randomly chosen to be discarded. Note that although the random drop scheme fixes this problem, it cannot alleviate the other two fairness problems discussed in the next sections.

ARB: Now, Let us investigate the simulation results of ARB using the same configuration. As shown in figure 13, ARB makes a great improvement compared with TCP-SS. The percentage of throughput of node 1 for various RTT ratios is around 50% with a small ripple. In fact, when we run a longer simulation time (3600 seconds), each connection equally shares the bandwidth of the bottleneck.

Unlike TCP-SS using a nonpaced window scheme, ARB paces out packets by a local timer. The deterministic relationship between packet arrivals and departures at the gateway is invalid in ARB. Therefore, it is not possible to build up the arrival pattern at a gateway mentioned in TCP-SS. In addition, the rate adjustment of ARB, an additive increase and multiplicative decrease, is the key to connections fairly sharing the bandwidth of the bottleneck.

4.3: Bias against a Connection with Long RTT

TCP-SS: Using the same configuration in Figure 5, the maximum window size of each source is equal to the delay-bandwidth product. Node 1's round trip time is fixed with the value corresponding to pipeline size equal to 22 packets. Node 2's round trip time is adjusted up to 8 times node 1's. The simulation result is shown in figure 14. The x-axis indicates the round trip time ratio, and the y-axis represents the ratio of throughput to bottleneck bandwidth. A solid line and a dashed line show node 1's throughput and node 2's throughput, respectively. Two gateway congestion schemes are investigated. The upper figure shows the result of the drop tail gateway, and the results of the random drop gateway is depicted in the lower figure. For each cluster of simulation, node 2's propagation delay is varied over 10 ms.
Figure 14 clearly reveals a strong bias against a connection with a longer round trip, node 2, for both the drop tail gateway and the random drop gateway. Note that, in the drop tail gateway, the throughput of node 1 is higher than node 2 when the ratio of RTT is around 1. This reflects the bias based on the arrival pattern as mentioned in section 4.2.

The bias against a connection with a longer RTT results from the TCP-SS window adjustment scheme. Recall the adaptive window scheme in TCP/SS. The window size increases roughly by one packet per RTT, in a congestion avoidance phase. In addition, the throughput of TCP is equal to $\frac{W}{RTT}$. That is the increase rate of throughput is a function of $\frac{1}{RTT^2}$. Therefore, a connection with a longer RTT will take a significant time to recover its throughout after it detects a dropped

packet. Obviously, the random drop gateway cannot alleviate the fairness due to the different RTT.

ARB: Applying the same configurations, we show the simulation result of ARB for various RTT ratios in figure 15. The solid line and dot line represent the node 1's and node 2's throughput, respectively. A remarkable improvement is clearly shown in these figures. Throughput is not a function of the value of RTT. The two connections share the bandwidth almost equally.

The reasons for this outstanding fairness performance are twofold. First, as mentioned before, the rate adjustment is based on an additive increase and multiplicative decrease scheme. This mechanism has been shown in [3] to satisfy the goals of both fairness and efficiency. Second, we use an implicit feedback rate as a feedback signal to adjust the sending rate (throughput). This feedback rate, computed at the receiver, is independent of its round trip delay.

4.4: Bias against Bursty Traffic

TCP-SS: Here, bursty traffic means that the packets arrive at the gateway with a long interarrival time following many smaller interarrival times. In the simulation, bursty traffic is generated by a connection with a long round trip delay and a small window size. Since TCP-SS intends to send an entire window's worth of packets together, the connection will send a burst of packets and wait for a rough RTT to send the next burst.

The configuration in Figure 5 is extended so that there are 4 short FTP connections and one long FTP connection as shown in Figure 6 below. Each connection has a maximum window size of 8. Note that, in this one-way traffic configuration, the throughput of each connection should be limited by (Maximum Window Size)/RTT. That is the throughput of the bursty connections should consume around 6.5% of the total bandwidth at the router.

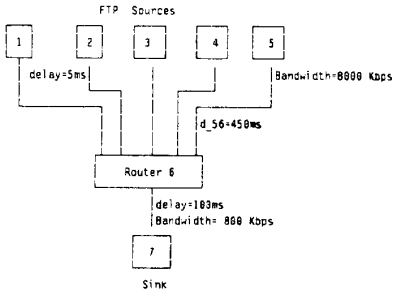

Figure 5. Simulation Network Configuration for Bursty Traffic.

Figure 16 and figure 17 show node 5's throughput (in percentage of total bandwidth) versus the maximum queue size at the gateway, with d_{56} = 449.4 msec and 453 msec, respectively. The dash line and solid line represent the average throughput of node 5 using a drop tail gateway and a random drop gateway, respectively. Each mark indicates the average throughput of a 50-sec period, excluding the first 50 second, a transit period.

The figures reveal that TCP-SS has a bias against bursty traffic when the buffer size is small, less than 14 packets. This is because the queue is more likely to overflow when a cluster of packets sent by the bursty traffic source hits the gateway. That is the packets from node 5 incur a disproportionate probability of being dropped. For example, with d_{56}=453 msec, node 5 obtains from 1% to 4% of the bandwidth, and incurs up to 100% of the packet drops. The throughput of bursty traffic is very sensitive to a small change in maximum buffer size at the gateway. In addition, with a drop tail gateway, throughput is more sensitive to the propagation delay. On the other hand, with a random drop gateway, the figures shows that the performance of node 5 is less sensitive to the propagation delay. However, it cannot cure the fairness problems either. For example, with d_{56}=453 msec, node 5 obtains from 1% to 5% of the bandwidth, and gets around 20% of the packet drops. This is because the queue contents at the time of overflows do not reflect the average traffic through the queue.

ARB: We have shown the fairness problem of bursty traffic in TCP-SS results from the clustering of packets in bursty traffic. In ARB, we use two mechanisms to greatly alleviate this problem. First, a congestion avoidance mechanism prevents offered traffic exceeding the capacity of the bottleneck. Therefore, we will rarely find a packet dropped at the gateway. Second, we apply a rate-base mechanism to pace out the packets into the network. With this mechanism, the arrivals of packet at the gateway is always proportional to the average rate through the queue. On the other hand, a cluster of packets from the same connection will be eliminated.

Recall the simulation results in section 3, the queue length at the gateway stays in a small region. This implies that clusters of packets do not hit the gateway. Although we do not show simulation results for ARB for the bursty traffic case, from the above two arguments and the simulation results in section 3, we believe that the evidences suffices to support our claim. That is ARB does not have the bias against bursty traffic that is present in TCP-SS.

5: Conclusions

In this report, we compared the performance of ARB and TCP-SS. The comparisons focus on the congestion control function in these two protocols, in which ARB adopts a congestion prevention mechanism with an adaptive rate-based scheme, and TCP-SS implements a congestion reaction mechanism with an adaptive window-scheme. We break the comparisons into two major areas: 1) Network efficiency, 2) Fairness. The comparison results clearly show that ARB outperforms TCP-SS in these two areas. In short, TCP-SS suffers from: an oscillating behavior in queue length, congestion window size and packet delay time; periodic packet losses; a significant underutilization of the transmission line; systematic discrimination of a particular connection in periodic traffic; bias against a connection with a longer round trip time; and bias against bursty traffic.
The problems that TCP-SS suffers are caused by the nature of the TCP-SS mechanism, mainly it: 1) uses a packet-based acknowledgement scheme, 2) adopts a congestion reaction mechanism, 3) uses non-paced window scheme, 4) relies on ACK arrivals providing a reliable clocking function, 5) uses an increase-by-one window adjustment algorithm.

Note that the phenomena described in this report are not only exhibited in TCP-SS but also shared by any congestion control algorithm having the above features. In fact, in [11], the measurements of an OSI testbed network revealed similar unfairness phenomena in two-way traffic. In this study, OSI/TP4 enhanced with the CE-bit congestion avoidance scheme was implemented. The theoretic explanations of this unfairness can be found in [6].

ARB can avoid the problems described above by: 1) adopting a congestion prevention mechanism, 2) using a rate-based scheme to pace out packets by a local timer, 3) controlling the sending rate by an explicit feedback rate from the receiver, and 4) using an additive increase and multiplicative decrease scheme to adjust the sending rate.

Simulation results of ARB in other configurations besides those presented here [7] have also shown that ARB can achieve good overall network performance as shown in this paper. Work is also being conducted to investigate the use of ARB in controlling traffic accessing an ATM network.

Acknowledgements: we would like to thank ACM and Sally Floyd for their permissions to reuse the figures in the papers cited.

References:

[1] R. Braden (editor), "Requirements for Internet Hosts-communication Layer," RFC-1122, Oct. 1989.

[2] D. Clark et. al., "An Analysis of TCP Processing Overhead," *IEEE Com. Magazine*, 6/1989.

[3] D. Chiu, R. Jain, "Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks," *Computer Networks and ISDN System*, vol.17, 1989, pp.1-14.

[4] S. Floyd, V. Jacobson, "Traffic Phase Effect in Packet-Switched Gateways," *ACM Computer Communication Review*, 21(2), 4/1991.

[5] S. Floyd, "Connections with Multiple Congested Gateways in Packet-Switched Networks Part 1: One-Way Traffic," *ACM Computer Communication Review*, 21(5), 10/1991.

[6] S. Floyd, V. Jacobson, "On Traffic Phase Effects in Packet-Switched Gateways," to appear in *Internetworking: Research and Experience*.

[7] L. Huynh, et. al., "ARB: An Adaptive Rate-Based Flow/Congestion Management for High-Speed Networks," *In preparation*

[8] V. Jacobson, "Congestion Avoidance and Control," *ACM SIGCOMM'88*, pp. 314-329, 8/1988.

[9] A. Mankin, K. Ramakrishnan (editors), "Gateway Congestion Control Survey," RFC 1254, 8/1991.

[10] S. Shenker, L. Zhang, D. Clark, "Some Observations on the Dynamics of a Congestion Control Algorithm," *ACM Computer Comm. Rev*, 20(4), 10/1990.

[11] R. Wilder, K. Ramakrishnan, A. Mankin, "Dynamics of a Congestion Control and Avoidance of Two-Way Traffic in OSI Testbed," *ACM Computer Communication Review*, 21(2), 4/1991.

[12] L. Zhang, S. Shenker, D. Clark, "Observations on the Dynamics of a Congestion Control Algorithm: The Effects of Two-Way Traffic," *ACM SIGCOMM'91*, 9/1991.

[13] R. Jain, "Myths About Congestion Management in High-Speed Networks" *Information Network and Data Communication IV. Elsevier Science Publishers B.V. (North-Holland)*, 1992.

[14] N. F. Maxemchuk, M. E. Zarki, "Routing and Flow Control in High-Speed Wide-Area Networks" *Proceedings of the IEEE*, Vol 78, No 1, 1/1990.

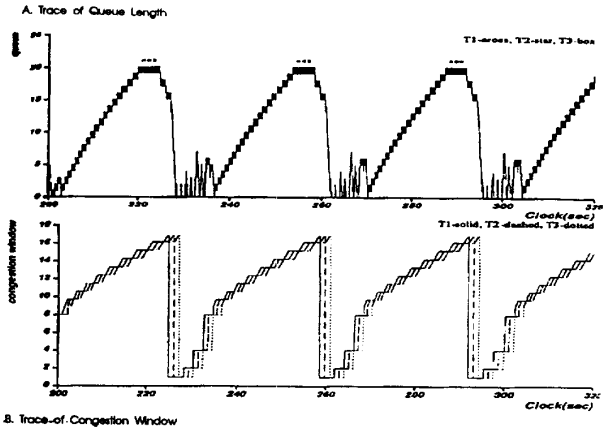

Figure 6. TCP one-way traffic.

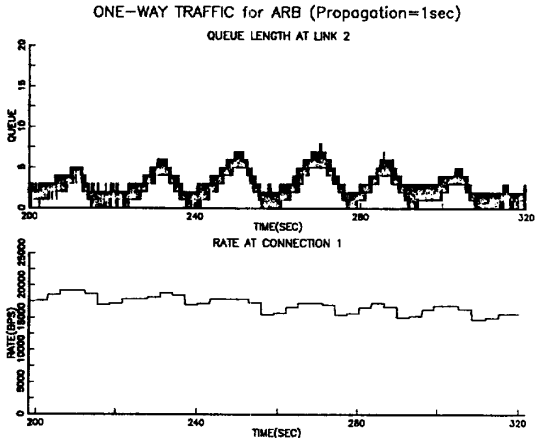

Figure 7. ARB one-way traffic.

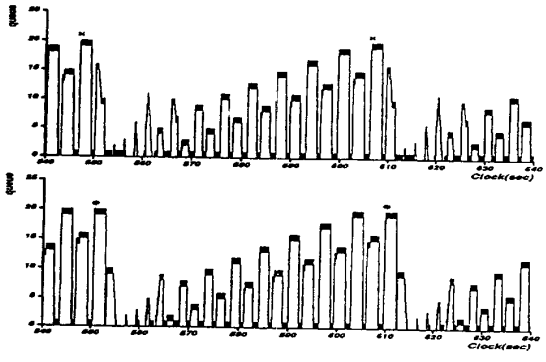

Figure 8. TCP two-way traffic for large pipeline.

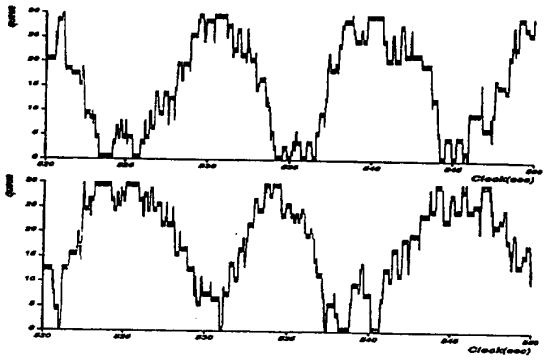

Figure 9. TCP two-way traffic for small pipeline.

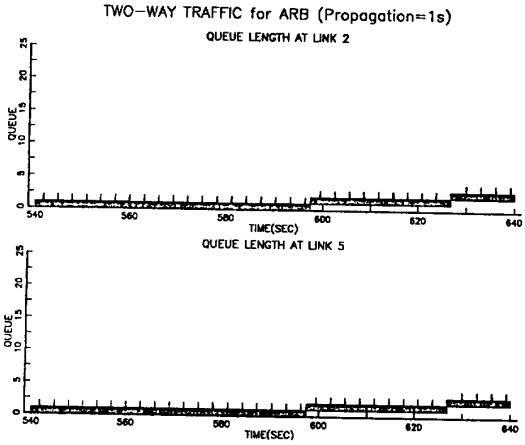

Figure 10. ARB two-way traffic for a large pipeline.

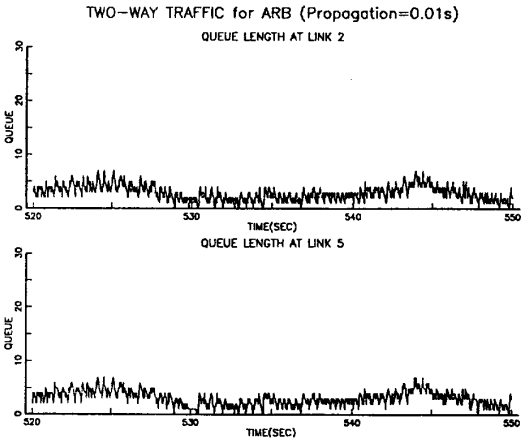

Figure 11. ARB two-way traffic for a small pipeline.

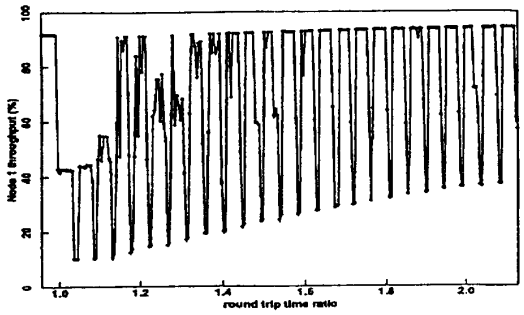

Figure 12. Bias in a periodic traffic in TCP/SS.

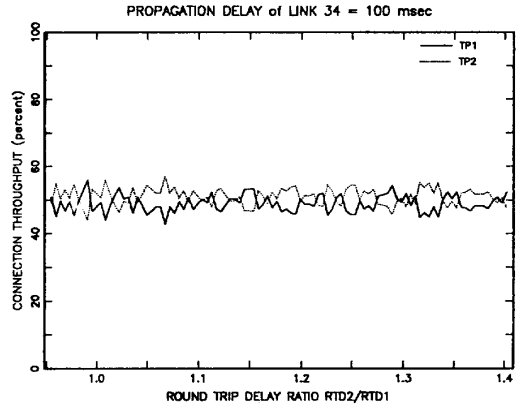

Figure 13. Fairness of periodic traffic in ARB.

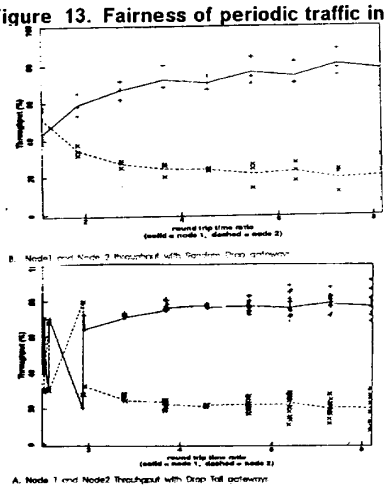

Figure 14. Bias against long RTT connection in TCP-SS.

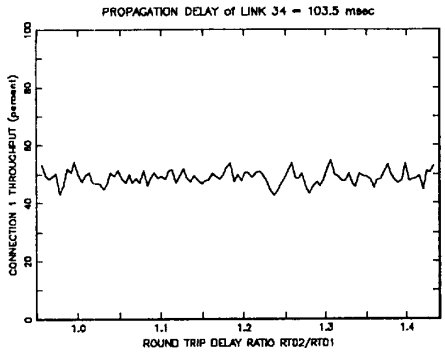

Figure 15. Fairness of a connection with long RTT in ARB.

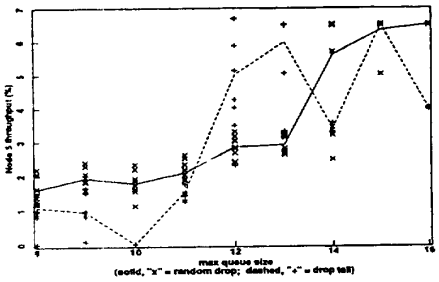

Figure 16. Bias against bursty traffic in TCP-SS. d_{56} = 449.3 msec.

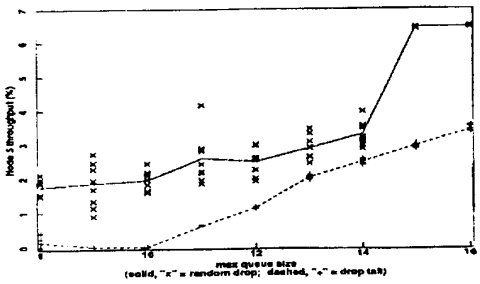

Figure 17. Bias against Bursty Traffic in TCP/SS. d_{56} = 453 msec