****
ATM Forum Document Number: ATM_Forum/98-0876R1
****

TITLE: Performance Analysis of TCP Enhancements for WWW Traffic using UBR+ with Limited Buffers over Satellite Links

****

SOURCE: Mukul Goyal, Rohit Goyal, Raj Jain, Bobby Vandalore, Sonia Fahmy
The Ohio State University,

Raj Jain is now at
Washington University in Saint Louis
Jain@cse.wustl.edu
http://www.cse.wustl.edu/~jain/

ation Science,
OH 43210

Thomas vonDeak and Kul Bhasin
NASA Lewis Research Center, MS: 54-6
21000 Brookpark Road
Cleveland, OH 44135
Email: {tvondeak, bhasin}@lerc.nasa.gov

Norm Butts and Sastri Kota
Lockheed Martin Telecommunications
1272 Borregas Ave
Sunnyvale, CA 94089
Email: {Norm.Butts, Sastri.Kota}@lmco.com

****

DISTRIBUTION: ATM Forum Technical Committee
Traffic Management Working Group

****

DATE: December, 1998 (Nashville)

****

ABSTRACT: This contribution presents the simulation results on the performance of TCP enhancements over ATM-UBR+ for satellite latencies with limited switch buffers for WWW traffic. We analyze the impact of Fast Retransmit and Recovery (FRR or Reno), New Reno, Selective Acknowledgement (SACK), Early packet drop (EPD), and Selective Drop on LEO, MEO, and GEO satellite configurations for several buffer sizes.
****

****

1 Introduction

In recent years there has been a tremendous growth in the amount of WWW traffic over the Internet. WWW traffic is essentially bursty in nature with periods of activity. The traffic pattern generated by a large number of WWW connections is expected to be different from that generated by persistent TCP traffic. In this contribution, we study the performance of WWW traffic over long delay networks for the UBR+ service. In our previous work [GOYAL97a, GOYAL97b, KOTA97], we have assessed the performance of persistent TCP traffic over UBR+ for various TCP options, UBR+ drop policies, buffer sizes and link delays. In this contribution, we extend the previous studies to WWW traffic. We also introduce another TCP model, NewReno [HOE96][FALL96], into our suite of TCP enhancements.

In this study, using our WWW traffic model, we perform full factorial simulations [JAIN91] involving

- **TCP flavors**: Vanilla, Fast Retransmit Recovery (Reno), NewReno and SACK

- **UBR+ drop policies**: Early Packet Drop (EPD) and Selective Drop (SD)

- **Propagation delays**: Satellite (Single-hop GEO, multiple-hop LEO/single-hop MEO) and WAN delays

- **Buffer sizes**: We use three buffer sizes approximately corresponding to 0.5, 1, and 2 times the round trip delay-bandwidth products

The simulation results are analyzed using ANOVA techniques presented in [JAIN91], and briefly described in Section 8.

Section 2 briefly discusses the key results of our previous work in TCP over UBR+. Section 3 presents an overview of the TCP enhancements, especially NewReno and SACK. Section 4 briefly overviews two UBR+ drop policies, EPD and Selective Drop, used in our simulations. We then describe our WWW model in Section 5. Finally, we present our simulation experiments, techniques, performance metrics, results and analysis.

2 Previous Work

In our past work, we have studied the performance of TCP over UBR+ and GFR for persistent TCP traffic. Studies have shown that TCP results in poor performance over UBR. Performance was measured in terms of efficiency and fairness. TCP performance over UBR can be improved by enhanced end-system policies as well as switch buffer management policies (or drop policies). The results for **persistent TCP** over UBR+ can be summarized as follows [GOYAL97a][GOYAL97b][GOYAL98]:

- TCP performance over UBR can be improved by TCP enhancements, intelligent drop policies, guaranteed rate and sufficient buffer sizing.

- For low delay networks, intelligent drop policies provide the most improvement in performance.

- For long delay networks, end system policies have a more significant effect than drop policies. TCP SACK results in the best performance[1].

- Providing a guaranteed rate to the UBR service categories significantly improves the performance of TCP over UBR+ especially for long delay networks.

---

[1] In our previous work, we did not assess the performance of NewReno by simulation.

- For satellite networks with Selective Drop and TCP SACK, a buffer size of 0.5 RTT*bandwidth is sufficient for high performance even for a large number of TCP sources. Although higher buffer sizes improve the performance, the improvement is small. Lower buffer sizes decrease the performance significantly (both efficiency and fairness).

3 TCP Enhancements

TCP uses a window-based flow control and uses it also to limit the number of segments in the network. "Vanilla TCP" consists of the slow start and congestion avoidance phases for congestion control. It detects segment losses by the retransmission timeout. Coarse granularity of timeouts are the primary cause of low TCP throughput over the UBR service. TCP Reno implements the fast retransmit and recovery algorithms that enable the connection to quickly recover from isolated segment losses [STEV97]. TCP Reno can efficiently recover from isolated segment losses, but not from bursty losses. As a result, TCP Reno results in poor efficiency for long latency configurations especially for low buffer sizes and persistent traffic.

3.1 TCP New Reno: A Modification to Fast Retransmit and Recovery

As indicated above, TCP Reno can not recover effectively from multiple packet drops. A modification to Reno, popularly known as NewReno was proposed by Jenny Hoe [HOE96] to overcome this shortcoming. She introduced a "fast-retransmit phase", in which the sender remembers the highest sequence number sent (RECOVER) when the fast retransmit is first triggered. After the first unacknowledged packet is retransmitted (when three duplicate ACKs are received), the sender follows the usual fast recovery algorithm and increases the CWND by one segment for each duplicate ACK it receives. When the sender receives an acknowledgment for the retransmitted packet, it checks if the ACK acknowledges all segments including RECOVER. If so, the sender exits the fast retransmit phase, sets its CWND to SSTHRESH and starts a linear increase (congestion avoidance phase). On the other hand, if the ACK is a partial ACK, i.e., it acknowledges the retransmitted segment and only some of the segments before RECOVER, then the sender immediately retransmits the next expected segment as indicated by the ACK. A partial ACK also resets the CWND to SSTHRESH. This continues until all segments including RECOVER are acknowledged. The NewReno retransmits one segment every round trip time until an ACK is received for RECOVER. This mechanism ensures that the sender will recover from N segment losses in N round trips.

Very recently, a description of the NewReno algorithm has appeared in [FLOY98]. This description recommends a modification in which on receiving a partial ACK the congestion window is reduced by amount of new data acknowledged and then incremented by 1 MSS. Another modification is suggested to avoid multiple fast retransmits. Our implementation of NewReno reflects the behavior as implemented in version ns-2.1b3 of *ns* simulator [NS] and does not have these modifications.

Another issue raised in [FLOY98] is whether the retransmit timer should be reset after each partial ACK or only after the first partial ACK. For satellite links, where retransmission timeout value is not much larger than the round trip time (RTT), the first option is better. If the retransmit timer is reset only after the first partial ACK, a retransmission timeout will be caused even for a small number of packets lost in a window. For satellite links with their long delays, a timeout is very costly. However, for WAN links, the retransmission timeout value is much larger than the RTT. For WAN links, if there are a number of packets lost in a window, it is better to timeout and retransmit all the packets using slow-start than to retransmit just 1 packet every RTT. In such a case, the second option is better. In our implementation, we reset the retransmit timer after each partial ACK.

Further, in our implementation of NewReno, we have incorporated two changes suggested by [HOE96]. The first suggestion is to set the initial SSTHRESH value to RTT-bandwidth product. The second is to send one new packet beyond RECOVER upon receiving 2 duplicate ACKs while in the fast-retransmit

phase (to keep the "flywheel" going). In our implementation, on receiving a partial ACK, a single packet is retransmitted. Since the TCP delay ACK timer is NOT set, all segments are ACKed as soon as they are received.

3.2 SACK TCP: Selective Acknowledgements

TCP with Selective Acknowledgments (SACK TCP) has been proposed to efficiently recover from multiple segment losses [MATH96]. In SACK TCP, acknowledgments contain additional information about the segments that have been received by the destination. When the destination receives out-of-order segments, it sends duplicate ACKs (SACKs) acknowledging the out-of-order segments it has received. From these SACKs, the sending TCP can reconstruct information about the segments not received at the destination. On receiving three duplicate ACKs, the sender retransmits the first lost segment and enters "fast-retransmit" phase as in NewReno. The CWND is set to half its current value. SSTHRESH is set to the new value of CWND and the highest sequence number sent so far is recorded in RECOVER. As in NewReno, the sender does not come out of the fast-retransmit phase until it has received the ACK for RECOVER. However, in the fast-retransmit phase if allowed by the window, the sending TCP uses the SACK information to retransmit lost segments before sending any new data. A sender implementing NewReno can retransmit only one lost segment every RTT. Thus it recovers from N segment losses in N RTTs. However, a sender implementing SACK can recover from N segment losses much faster.

Our implementation of SACK is based on the description in [FALL96][FLOY95][MATH96]. The SACK is sent whenever out-of-sequence data is received. All duplicate ACKs contain the SACK option. The receiver keeps track of all the out-of-sequence data blocks received. When the receiver generates a SACK, the first SACK block specifies the block of data formed by the most recently received data segment. This ensures that the receiver provides the most up to date information to the sender. After the first SACK block, the remaining blocks can be filled in any order.

The sender keeps a table of all the segments sent but not ACKed. When a segment is sent, it is entered into the table. When the sender receives an ACK with the SACK option, it marks all the segments specified in the SACK option blocks as SACKed. The entries for each segment remain in the table until the segment is ACKed. When the sender receives three duplicate ACKs, it retransmits the first unacknowledged packet and enters fast-retransmit phase. During the fast retransmit phase, when the sender is allowed to send, it first tries to retransmit the holes in the SACK blocks before sending any new segments. When the sender retransmits a segment, it marks the segment as retransmitted in the table. If a retransmitted segment is lost, the sender times out and performs slow start. When a timeout occurs, the sender resets the SACK table.

During the fast retransmit phase, the sender maintains a variable called "PIPE" that indicates the number of bytes currently in the network. When the third duplicate ACK is received, PIPE is set to the value of CWND - 3 segments and CWND is reduced by half. For every subsequent duplicate ACK received, PIPE is decremented by one segment because the ACK denotes a packet leaving the network. The sender sends data (new or retransmitted) only when PIPE is less than CWND value. This implementation is equivalent to inflating the CWND by one segment for every duplicate ACK and sending segments if the number of unacknowledged bytes is less than the congestion window value.

When a segment is sent, PIPE is incremented by one segment. When a partial ACK is received, PIPE is decremented by two. The first decrement is because the partial ACK represents a retransmitted segment leaving the pipe. The second decrement is done because the original segment that was lost, and had not been accounted for, is now actually considered to be lost.

4 UBR+ Drop Policies

The basic UBR service can be enhanced by implementing intelligent drop policies at the switches. In this study, we have used EPD and Selective Drop as drop policies.

4.1 Early Packet Discard (EPD)

The Early Packet Discard policy [ROMA95] maintains a threshold R, in the switch buffer. When the buffer occupancy exceeds R, then all new incoming packets are dropped. Partially received packets are accepted if possible.

The drop threshold R should be chosen so that on crossing the threshold, the switch has enough buffer left to accept cells of all incomplete packets currently in the buffer. However, if the number of VCs passing through the switch is large, it is possible that the entire buffer may not be enough to store one complete packet of each VC.

4.2 Selective Drop (SD)

The Selective Drop policy [GOYAL98] uses per-VC accounting, i.e., keeps track of current buffer utilization of each active UBR VC. A UBR VC is called "active" if it has at least one cell currently buffered in the switch. The total buffer occupancy, X, is allowed to grow until it reaches a threshold R, maintained as a fraction of the buffer capacity K. A fair allocation is calculated for each active VC, and if the VC's buffer occupancy X_i exceeds its fair allocation, its subsequent incoming packet is dropped. Mathematically, in the Selective Drop scheme, an active VC's entire packet is dropped if

$$(X > R) \text{ AND } (X_i > Z \ X/N_a)$$

where N_a is the number of active VCs and Z is another threshold parameter ($0 < Z <= 1$) used to scale the effective drop threshold.

5 WWW Traffic Model

The WWW uses Hypertext Transfer Protocol (HTTP). HTTP uses TCP/IP for communication between WWW clients and WWW servers [LEE96]. Modeling of the WWW traffic is difficult because of the changing nature of web traffic. In this section, we outline our model and the inherent assumptions.

5.1 Implications of the HTTP/1.1 standard

The main difference between version 1.1 of the Hypertext Transfer Protocol, HTTP/1.1 [FIEL97], and earlier versions is the use of persistent TCP connections as the default behavior for all HTTP connections. In other words, a new TCP connection is not set up for each HTTP/1.1 request. The HTTP client and the HTTP server assume that the TCP connection is persistent until a *Close* request is sent in the HTTP Connection header field.

Another important difference between HTTP/1.1 and earlier versions is that the HTTP client can make multiple requests without waiting for responses from the server (called *pipelining*). Earlier models were *closed-loop* in the sense that each request needed a response before the next request could be sent.

5.2 WWW Server Model

Our WWW traffic arrival model is an extension of that specified in SPECweb96 benchmark [SPEC96]. In our model, a WWW server, on receiving a request from a WWW client sends some data back. The amount of data to be sent (the requested file size) is determined by classifying the client request into one of five classes (Class 0 through Class 4), shown in Table 1. As shown in the table, 20% of the requests are classified as Class 0 requests, i.e., less than 1 KB of data is sent in the response. Similarly 28% of file requests are classified as Class 1 requests and so on. We model our WWW servers as infinitely fast servers, i.e., the response is generated as soon as the request is received.

Table 1 WWW Server File Size Classes

Class	File Size Range	Frequency Of Access
Class 0	0 - 1 KB	20 %
Class 1	1 KB - 10 KB	28 %
Class 2	10 KB – 100 KB	40 %
Class 3	100 KB - 1 MB	11.2 %
Class 4	1 MB - 10 MB	0.8 %

There are nine discrete sizes in each class (e.g. Class 1 has 1 KB, 2 KB, up to 9 KB and Class 2 has 10 KB, 20 KB, through 90 KB and so on.). Within a class, one of these nine file sizes is selected according to a Poisson Distribution with mean 5. The model of discrete sizes in each class is based on the SPECweb96 benchmark [SPEC96]. The key differences from the SPEC model are

(i)   the assumptions of an infinite server, i.e. no processing time taken by server for a client request, and

(ii)   a new class of file sizes (1 MB - 10 MB), which allows us to model file sizes larger than those in the SPEC benchmark and the corresponding change in the percentage distribution of client requests into server file size classes.

The reason for the new class of file sizes is to model downloading of large software and offline browsing of search results. The percentages of requests falling into each of file size classes have been changed so that average requested file size is around 120 KB, as opposed to 15 KB in SPECweb96 model. We believe the new figure better represents the current WWW traffic scenario. The reason for having 20% of the requests classified as Class 0 requests is explained in next sub-section.

5.3 WWW Client Model

The HTTP-model in [MAH97] describes an empirical model of WWW clients based on observations in a LAN environment. Specifically, a typical client is observed to make, on the average, four HTTP GET requests for a single document. Multiple requests are needed to fetch inline images, if any. With the introduction of JAVA scripts in web pages, additional accesses maybe required to fetch the scripts. Therefore, we use five as the average number of HTTP GET requests. In our model, a WWW client makes 1 to 9 requests for a single document, The number is Poisson distributed around a mean of 5.

These requests are separated by a random time interval between 100 ms to 500 ms. Caching effects at the clients are ignored.

Typically, the first request from an HTTP client accesses the index page (plain text), which is of size 1 KB or less. Since every fifth request is expected to be an index page access, WWW server classifies 20% (= 1/5) of the client requests as Class 0 requests and sends 1 KB or less data in the response.

We also model a time lag between batches of requests (presumably for the same document), that corresponds to the time taken by the user to request a new document, as a constant, 10 seconds. While this may be too short a time for a human user to make decisions, it also weights the possibility of offline browsing where the inter-batch time is much shorter.

We do not attempt to model user behavior across different servers. The main purpose of using this simplistic model is to approximate the small loads offered by individual web connections, and to study the effects of aggregation of such small loads on the network.

6 Simulation Experiments

Figure 1 shows the configuration used in all our simulations. The configuration consists of 100 WWW clients being served by 100 WWW servers, one server for each client. Both WWW clients and servers use underlying TCP connections for data transfer. The switches implement the UBR+ service with optional drop policies described earlier. The following subsections describe various configuration parameters, TCP parameters, and switch parameters used in the simulations.

Figure 1 Simulation Configuration with 100 WWW Client-Server Connections

6.1 Configuration Parameters

- Links connecting server/client TCPs to switches have a bandwidth of 155.52 Mbps (149.76 Mbps after SONET overhead), and a one way delay of 5 microseconds.

- The link connecting the two switches has a bandwidth of 45Mbps (T3). It simulates one of three scenarios: a WAN, a multiple hop LEO/single hop MEO or a GEO link. The corresponding one-way link delays are 5 ms, 100 ms and 275 ms, respectively.

- Since the propagation delays on the links connecting client/server TCPs to switches are negligible compared to the delay on the inter-switch link, the round trip times (RTTs) due to propagation delay are 10 ms, 200 ms and 550 ms for WAN, LEO/MEO and GEO, respectively.

- All simulations run for 100 secs. Since every client makes a new set of requests every 10 secs, the simulations run for 10 cycles of client requests.

6.2 TCP Parameters

- Underlying TCP connections send data as specified by the client/server applications. A WWW client asks its TCP to send a 128 byte packet as a request to the WWW server TCP.

- TCP maximum segment size (MSS) is set to 1024 for WAN links and 9180 for LEO/MEO and GEO links.

- TCP timer granularity is set to 100 ms.

- TCP maximum receiver window size is chosen so that it is always greater than RTT-bandwidth product of the path. Such a value of receiver window ensures that receiver window does not prevent sending TCPs from filling up the network pipe. For WAN links (10 ms RTT due to propagation delay), the default receiver window size of 64K is sufficient. For MEO links (200 ms RTT), RTT-bandwidth product is 1,125,000 bytes. By using the TCP window scaling option and having a window scale factor of 5, we achieve an effective receiver window of 2,097,120 bytes. Similarly, for GEO links (550 ms RTT), the RTT-bandwidth product is 3,093,750 bytes. We use a window scale factor of 6 to achieve an effective receiver window of 4,194,240 bytes.

- TCP "Silly Window Syndrome Avoidance" is disabled because in WWW traffic many small segments (due to small request sizes, small file sizes or last segment of a file) have to be sent immediately.

- It has been proposed in [HOE96] that instead of having a fixed initial SSTHRESH of 64 KB, the RTT-bandwidth product of the path should be used as initial SSTHRESH. In our simulations, we have implemented this. Hence, the initial SSTHRESH values for WAN, MEO and GEO links are 56,250, 1,125,000 and 3,093,750 bytes respectively.

- The TCP delay ACK timer is NOT set. Segments are ACKed as soon as they are received.

6.3 Switch Parameters

- The drop threshold R is 0.8 for both drop policies - EPD and SD. For SD simulations, threshold Z also has a value 0.8.

- We use three different values of buffer sizes in our experiments. These buffer sizes approximately correspond to 0.5, 1, and 2 RTT - bandwidth products. For WAN delays, the largest buffer size is 2300 cells. This is a little more than the 2 RTT - bandwidth product. The reason for selecting 2300 is that this is the smallest buffer size that can hold one complete packet (MSS=1024 bytes) for each of the 100 TCP connections. For WAN, 0.5 RTT and 1 RTT buffers are not sufficient to hold one packet from each of the 100 TCPs. This problem will also occur in MEO and GEO TCPs if the number of TCPs is increased. Some preliminary analysis has shown that the buffer size required for good performance may be related to the number of active TCP connections as well as the RTT-bandwidth product. Further research needs to be performed to provide conclusive results of this effect. Table 2 shows the switch buffer sizes used in the simulations.

Table 2 Switch Buffer Sizes used for Simulations

Link Type (RTT)	RTT-bandwidth product (cells)	Switch Buffer Sizes (cells)
WAN (10 ms)	1062	531, 1062, 2300
Multiple-Hop LEO/Single-Hop MEO (200 ms)	21230	10615, 21230, 42460
Single-Hop GEO (550 ms)	58380	29190, 58380, 116760

7 Performance Metrics

The performance of TCP is measured by the efficiency and fairness index which are defined as follows. Let x_i be the throughput of the *i*th TCP connection $(1 \leq i \leq 100)$. Let *C* be the maximum TCP throughput achievable on the link. Let E be the efficiency of the network. Then, *E* is defined as

$$E = \frac{\sum_{i=1}^{i=N} x_i}{C}$$

where N = 100 and $\sum x_i$ is sum of all 100 server throughputs.

The TCP throughput values are measured at the client TCP layers. Throughput is defined as the highest sequence number in bytes received at the client from the server divided by the total simulation time. The results are reported in Mbps.

Due to overheads imposed by TCP, IP, LLC and AAL5 layers, the maximum possible TCP over UBR throughput depends on the TCP maximum segment size (MSS). For MSS = 1024 bytes (on WAN links), the ATM layer receives 1024 bytes of data + 20 bytes of TCP header + 20 bytes of IP header + 8 bytes of LLC header + 8 bytes of AAL5 trailer. These are padded to produce 23 ATM cells. Thus, each TCP segment of 1024 bytes results in 1219 bytes at the ATM layer. Thus, the maximum possible TCP throughput *C* is 1024/1219 = 84%. This results in 37.80 Mbps approximately on a 45 Mbps link. Similarly, for MSS = 9180 bytes (on MEO, GEO links), *C* is 40.39 Mbps approximately. Since, the "Silly Window Syndrom Avoidance" is disabled (because of WWW traffic), some of the packets have less than 1 MSS of data. This decreases the value of *C* a little. However, the resulting decrease in the value of *C* has an insignificant effect on the overall efficiency metric.

In all simulations, the 45 Mbps(T3) link between the two switches is the bottleneck. The average total load generated by 100 WWW servers is 48 Mbps[2].

---

[2] A WWW server gets on average 5 client requests every 10 s and sends on average 120 kB of data for each request. This means that on average a WWW server schedules 60 kBps i.e. 480 kbps of data. Hence average total load generated by 100 WWW servers is 48 Mbps.

We measure fairness by calculating the Fairness Index F defined by:

$$F = \frac{\left(\sum\limits_{i=1}^{i=N} x_i/e_i\right)^2}{N \times \sum\limits_{i=1}^{i=N} \left(x_i/e_i\right)^2}$$

where $N = 100$ and e_i is the expected throughput for connection i. In our simulations, e_i is the max-min fair share that should be allocated to server i. On a link with maximum possible throughput C, the fair share of each of the 100 servers is $C/100$. Let S_i be the maximum possible throughput that a server can achieve, calculated as the total data scheduled by the server for the client divided by simulation time.

For all i's for which $S_i < C/100$, $e_i = S_i$, i.e., servers that schedule less than their fair share are allocated their scheduled rates. This determines the first iteration of the max-min fairness calculation. These e_i's are subtracted from C, and the remaining capacity is again divided in a max-min manner among the remaining connections. This process is continued until all remaining servers schedule more than the fair share in that iteration, for those servers e_i = the fairshare.

8 Simulation Analysis and Results

In this section, we present a statistical analysis of simulation results for WAN, multiple hop LEO/single hop MEO and GEO links and draw conclusions about optimal choices for TCP flavor, switch buffer sizes and drop policy for these links. The analysis techniques we have used here are described in detail in [JAIN91]. The next subsection gives a brief description of these techniques. The following subsections present simulation results for WAN, LEO/MEO, and GEO links, respectively.

8.1 Analysis Technique

The purpose of analyzing results of a number of experiments is to calculate the individual effects of contributing factors and their interactions. These effects can also help us in drawing meaningful conclusions about the optimum values for different factors. In our case, we have to analyze the effects of the TCP flavors, buffer sizes and drop policies in determining the efficiency and fairness for WAN, MEO and GEO links. Thus, we have 3 factors: TCP flavor, switch buffer size and drop policy. The values a factor can take are called 'levels' of the factor. For example, EPD and SD are two levels of the factor 'Drop Policy'. Table 3 lists the factors and their levels used in our simulations.

<div align="center">Table 3 Factors and Levels in simulations</div>

Factor	Levels			
TCP flavor	Vanilla	Reno	NewReno	SACK
Switch drop policy	EPD		SD	
Switch buffer size	0.5 RTT[3]	1 RTT		2 RTT

---

[3] Here onwards, when we say 1 RTT worth of buffer, we mean a buffer size equal to the product of RTT and link bandwidth in terms of cells.

The analysis is done separately for efficiency and fairness, and consists of the calculating the following terms:

1. **Overall mean**: This consists of the calculation of the overall mean 'Y' of the result (efficiency or fairness).

2. **Total variation**: This represents the variation in the result values (efficiency or fairness) around the overall mean 'Y'..

3. **Main effects**: These are the individual contributions of a level of a factor to the overall result. A particular main effect is associated with a level of a factor, and indicates how much variation around the overall mean is caused by the level. We calculate the main effects of 4 TCP flavors, 3 buffer sizes, and 2 drop policies.

4. **First order interactions**: These are the interaction between levels of two factors. In our experiments, there are first order interactions between each TCP flavor and buffer size, between each drop policy and TCP flavor, and between each buffer size and drop policy.

5. **Allocation of variation**: This is used to explain how much each effect contributes to the total variation. An effect (a factor or interaction), which explains a large fraction of the total variation, is said to be important.

6. **Overall standard error**: This represents the experimental error associated with each result value. The overall standard error is also used in the calculation of the confidence intervals for each effect.

7. **Confidence intervals for main effects**: The 90% confidence intervals for each main effect are calculated. If a confidence interval contains 0, then the corresponding level of the factor is not statistically significant. If confidence intervals of two levels overlap, then the effects of both levels are assumed to be similar.

The first step of the analysis is the calculation of the overall mean 'Y' of all the values. The next step is the calculation of the individual contributions of each level 'a' of factor 'A', called the 'Main Effect'. The 'Main Effect' of 'A' at level 'a' is calculated by subtracting the overall mean 'Y' from the mean of all results with 'a' as the value for factor 'A'. The 'Main Effects' are calculated in this way for each level of each factor.

We then calculate the interactions between two factors. The interaction between levels of two factors is called 'First-order interaction'. For calculating the interaction between level 'a' of factor 'A' and level 'b' of factor 'B', an estimate is calculated for all results with 'a' and 'b' as values for factors 'A' and 'B'. This estimate is the sum of the overall mean 'Y' and the 'Main Effects' of levels 'a' and 'b'. This estimate is subtracted from the mean of all results with 'a' and 'b' as values for factors 'A' and 'B' to get the 'Interaction' between levels 'a' and 'b'. Although one could continue computing second and higher order interactions, we limit our analysis to 'First-order interactions' only. Higher order interactions are assumed to small or negligible.

We then perform the calculation of the 'Total Variation' and 'Allocation of Variation'. First, the value of the square of the overall mean 'Y' is multiplied by the total number of results. This value is subtracted from the sum of squares of individual results to get the 'Total Variation' among the results. The next step is the 'Allocation of Total Variation' to individual 'Main Effects' and 'First-order interactions'. To calculate the variation caused by a factor 'A', we take the sum of squares of the main effects of all levels of 'A' and multiply this sum with the number of experiments conducted with each level of 'A'. For example, to calculate the variation caused by TCP flavor, we take the sum of squares of the main effects of all its levels (Vanilla, Reno, NewReno and SACK) and multiply this sum by 6 (with each TCP flavor

we conduct 6 different simulations involving 3 buffer sizes and 2 drop policies). In this way, the variation caused by all factors is calculated. To calculate the variation caused by first-order interaction between two factors 'A' and 'B', we take the sum of squares of all the first-order interactions between levels of 'A' and 'B' and multiply this sum with the number of experiments conducted with each combination of levels of 'A' and 'B'.

The next step of the analysis is to calculate the overall standard error for the results. This value requires calculation of individual errors in results and the degrees of freedom for the errors. For each result value, an estimate is calculated by summing up the overall mean 'Y', main effects of the parameter levels for the result and their interactions. This estimate is subtracted from the actual result to get the error 'e_i' for the result.

If a factor 'A' has 'N_A' levels, then the total number of degrees of freedom is $\Pi(N_A)$. Thus, for our analysis, the total number of degrees of freedom is $4 \times 2 \times 3 = 24$. The degrees of freedom associated with the overall mean 'Y' is 1. The degrees of freedom associated with 'main effects' of a factor 'A' are '$N_A - 1$'. Thus, degrees of freedom associated with all 'main effects' are $\sum(N_A - 1)$. Similarly, the degrees of freedom associated with the first-order interaction between 2 factors 'A' and 'B' are $(N_A - 1) \times (N_B - 1)$. Thus, degrees of freedom associated with all first-order interactions are $\sum(N_A - 1) \times (N_B - 1)$, with the summation extending over all factors. In our analysis, the degrees of freedom associated with all 'main effects' are $3 + 1 + 2 = 6$ and the degrees of freedom associated with all first-order interactions are $(3 \times 1) + (3 \times 2) + (1 \times 2) = 11$.

Since we use the overall mean 'Y', the main effects of individual levels and their first-order interactions to calculate the estimate, the value of the degrees of freedom for errors 'd_e' is calculated as follows:

$$d_e = \prod(N_A) - 1 - \sum(N_A - 1) - \sum(N_A - 1) \times (N_B - 1)$$

In our case, $d_e = 24 - 1 - 6 - 11 = 6$.

To calculate the overall standard error 's_e', the sum of squares of all individual errors 'e_i' is divided by the number of degrees of freedom for errors 'd_e' (6 in our case). The square root of the resulting value is the overall standard error.

$$s_e = \sqrt{\left(\sum e_i^2\right) / d_e}$$

Finally, based on the overall standard error, we calculate the 90% confidence intervals for all 'main effects' of each factor. For this purpose, we calculate the standard deviation 's_A' associated with each factor 'A' as follows:

$$s_A = s_e \times \sqrt{(N_A - 1) / \prod(N_A)}$$

Here, 'N_A' is the number of levels for factor 'A' and $\Pi(N_A)$ is the total number of degrees of freedom.

The variation around the 'main effect' of all levels of a factor 'A' to get a 90% confidence level is given by the standard deviation 's_A' multiplied by $t[0.95,d_e]$, where $t[0.95,d_e]$ values are quantiles of the t distribution [JAIN91].

Hence, if 'ME_a' is the value of the main effect of level 'a' of factor 'A', then the 90% confidence interval for 'ME_a' is $\{ME_a \pm s_A \times t[0.95,d_e]\}$. The main effect value is statistically significant only if the confidence interval does not include 0.

This was only a brief description of the techniques used to analyze simulation results. The detailed description of the analysis method can be found in [JAIN91].

8.2 Simulation Results for WAN links

Table 4 presents the individual efficiency and fairness results for WAN links. Table 5 shows the calculation of 'Total Variation' in WAN results and 'Allocation of Variation' to main effects and first-order interactions. Table 6 shows the 90% confidence intervals for the main effects. A negative value of main effect implies that the corresponding level of the factor decreases the overall efficiency and vice versa. If a confidence interval encloses 0, the corresponding level of the factor is assumed to be not significant in determining performance.

Table 4 Simulation Results for WAN links

Drop Policy	TCP Flavor	Buffer = 0.5 RTT		Buffer = 1 RTT		Buffer = 2 RTT	
		Efficiency	Fairness	Efficiency	Fairness	Efficiency	Fairness
EPD	Vanilla	0.4245	0.5993	0.5741	0.9171	0.7234	0.9516
	Reno	0.6056	0.8031	0.7337	0.9373	0.8373	0.9666
	NewReno	0.8488	0.8928	0.8866	0.9323	0.8932	0.9720
	SACK	0.8144	0.7937	0.8948	0.8760	0.9080	0.8238
SD	Vanilla	0.4719	0.6996	0.6380	0.9296	0.8125	0.9688
	Reno	0.6474	0.8230	0.8043	0.9462	0.8674	0.9698
	NewReno	0.8101	0.9089	0.8645	0.9181	0.8808	0.9709
	SACK	0.7384	0.6536	0.8951	0.8508	0.9075	0.8989

Table 5 Allocation of Variation for WAN Efficiency and Fairness Values

Component	Sum of Squares		%age of Variation	
	Efficiency	Fairness	Efficiency	Fairness
Individual Values	14.6897	18.6266		
Overall Mean	14.2331	18.3816		
Total Variation	0.4565	0.2450	100	100
Main Effects:				
TCP Flavor	0.2625	0.0526	57.50	21.49
Buffer Size	0.1381	0.1312	30.24	53.55
Drop Policy	0.0016	0.0002	0.34	0.09
First-order Interactions:				
TCP Flavor-Buffer Size	0.0411	0.0424	8.99	17.32
TCP Flavor-Drop Policy	0.0104	0.0041	2.27	1.68
Buffer Size-Drop Policy	0.0015	0.0009	0.33	0.38
Standard Error, s_e = 0.0156(For Efficiency), 0.0472(For Fairness)				

Table 6 Main Effects and their Confidence Intervals for WAN

Factor	Main Effect		Confidence Interval	
	Efficiency	Fairness	Efficiency	Fairness
TCP Flavor:				
Vanilla	-0.1627	-0.0308	(-0.1734,-0.1520)	(-0.0632,0.0016)
Reno	-0.0208	0.0325	(-0.0315,-0.0101)	(0.0000, 0.0649)
NewReno	0.0939	0.0573	(0.0832,0.1046)	(0.0248, 0.0898)
SACK	0.0896	-0.0590	(0.0789,0.1003)	(-0.0914, -0.0265)
Buffer Size:				
0.5 RTT	-0.1000	-0.1034	(-0.1087,-0.0912)	(-0.1299,-0.0769)
1 RTT	0.0163	0.0382	(0.0076,0.0250)	(0.0117, 0.0647)
2 RTT cells	0.0837	0.0651	(0.0749,0.0924)	(0.0386, 0.0916)
Drop Policy:				
EPD	-0.0081	-0.0030	(-0.0142, -0.0019)	(-0.0217,0.0157)
SD	0.0081	0.0030	(0.0019,0.0142)	(-0.0157, 0.0217)

8.2.1 Analysis of Efficiency values: Results and Observations

The following conclusions can be drawn from the above tables:

1. TCP flavor explains 57.5% of the variation and hence is the major factor in determining efficiency. It can be established from confidence intervals of effects of different TCP flavors that NewReno and SACK have better efficiency performance than Vanilla and Reno. Since the confidence intervals of effects of SACK and NewReno overlap, we cannot say that one performs better than the other. Confidence intervals for the effects of Vanilla and Reno suggest that Reno performs better than Vanilla.

2. Buffer size explains 30.24% of the variation and hence is the next major determinant of efficiency. Confidence intervals for effects of different buffer sizes clearly indicate that efficiency increases substantially as buffer size is increased. However, if we look at individual efficiency values, it can be noticed that only Vanilla and Reno get substantial increase in efficiency as buffer size is increased from 1 RTT to 2 RTT.

3. The interaction between buffer size and TCP flavor explains 8.99% of the variation. The large interaction is because of the fact that only Vanilla and Reno show substantial gains in efficiency as the buffer size is increased from 1 RTT to 2 RTT. For SACK and NewReno, increasing buffer sizes from 1 RTT to 2 RTT does not bring much increase in efficiency. This indicates that SACK and NewReno can tolerate the level of packet loss caused by a buffer size of 1 RTT.

4. Though the variation explained by drop policy is negligible, it can be seen that for Vanilla and Reno, SD results in better efficiency than EPD for the same buffer size. This is because for EPD, after crossing the threshold R, all new packets are dropped and buffer occupancy does not increase much beyond R. However for SD, packets of VCs with low buffer occupancy are still accepted. This allows the buffer to be utilized more efficiently and fairly and to better efficiency as well as fairness.

5. For NewReno and SACK, the efficiency values are similar for EPD and SD for same buffer size. This is because NewReno and SACK are much more tolerant of packet loss than Vanilla and Reno. Thus the small decrease in number of packets dropped due to increased buffer utilization does not cause a significant increase in efficiency.

6. It can be noticed from individual efficiency values that SACK generally performs a little better than NewReno except when buffer size is very low (0.5 RTT). Better performance of NewReno for very low buffer size can be explained as follows. Low buffer size means that a large number of packets are dropped. When in fast retransmit phase, NewReno retransmits a packet for every partial ACK received. However, SACK does not retransmit any packet till *PIPE* goes below *CWND* value. A large number of dropped packets mean that not many duplicate or partial ACKs are forthcoming. Hence *PIPE* may not reduce sufficiently to allow SACK to retransmit all the lost packets quickly. Thus, SACK's performance may perform worse than NewReno under extreme congestion.

We conclude that SACK and NewReno give best performance in terms of efficiency for WAN links. For NewReno and SACK, a buffer size of 1 RTT is sufficient for getting close to best efficiency with either EPD or SD as the switch drop policy.

8.2.2 Analysis of Fairness valu es: Results and Observations

1. Buffer size largely determines fairness as 53.55 % of the variation is explained by the buffer size. Confidence intervals for effects of buffer sizes suggest that the fairness increases substantially as buffer size is increased from 0.5 RTT to 1 RTT. Since confidence intervals for buffers of 1 RTT and 2 RTTs overlap, it cannot be concluded that 2 RTT buffers result in better performance than 1 RTT buffers.

2. TCP flavor is the next major factor in determining fairness as it explains 21.49 % of the variation. Confidence intervals for effects of TCP flavor on fairness, clearly suggest that NewReno results in the best fairness and SACK results in the worst fairness.

3. SD only increases fairness for low buffer sizes. Overall, both the allocation of variation to drop policy, and confidence intervals for effects of SD and EPD suggest that SD does not result in higher fairness when compared to EPD for bursty traffic in WAN links unless buffer sizes are small.

8.3 Simulation Results for MEO links

Table 7 presents the individual efficiency and fairness results for MEO links. Table 8 shows the calculation of 'Total Variation' in MEO results and 'Allocation of Variation' to main effects and first-order interactions. Table 9 shows the 90% confidence intervals for main effects.

Table 7 Simulation Results for MEO Links

Drop Policy	TCP Flavor	Buffer = 0.5 RTT		Buffer = 1 RTT		Buffer = 2 RTT	
		Efficiency	Fairness	Efficiency	Fairness	Efficiency	Fairness
EPD	Vanilla	0.8476	0.9656	0.8788	0.9646	0.8995	0.9594
	Reno	0.8937	0.9659	0.9032	0.9518	0.9091	0.9634
	NewReno	0.9028	0.9658	0.9105	0.9625	0.9122	0.9616
	SACK	0.9080	0.9517	0.9123	0.9429	0.9165	0.9487
SD	Vanilla	0.8358	0.9649	0.8719	0.9684	0.9009	0.9615
	Reno	0.8760	0.9688	0.8979	0.9686	0.9020	0.9580
	NewReno	0.8923	0.9665	0.8923	0.9504	0.8976	0.9560
	SACK	0.9167	0.9552	0.9258	0.9674	0.9373	0.9594

Table 8 Allocation of Variation for MEO Efficiency and Fairness Values

Component	Sum of Squares		%age of Variation	
	Efficiency	Fairness	Efficiency	Fairness
Individual Values	19.3453	22.1369		
Overall Mean	19.3334	22.1357		
Total Variation	0.0119	0.0012	100	100
Main Effects:				
TCP Flavor	0.0067	0.0003	56.75	29.20
Buffer Size	0.0026	0.0001	21.73	7.70
Drop Policy	0.0001	0.0001	0.80	6.02
First-order Interactions:				
TCP Flavor-Buffer Size	0.0016	0.0001	13.42	10.16
TCP Flavor-Drop Policy	0.0007	0.0003	6.11	22.60
Buffer Size-Drop Policy	0.0001	0.0001	0.53	6.03
Standard Error, s_e = 0.0036(For Efficiency), 0.0060(For Fairness)				

Table 9 Main Effects and Their Confidence Intervals for MEO

Factor	Mean Effect		Confidence Interval	
	Efficiency	Fairness	Efficiency	Fairness
TCP Flavor:				
Vanilla	-0.0251	0.0037	(-0.0276,-0.0226)	(-0.0004,0.0078)
Reno	-0.0005	0.0024	(-0.0030,0.0019)	(-0.0017,0.0065)
NewReno	0.0038	0.0001	(0.0013,0.0062)	(-0.0040,0.0042)
SACK	0.0219	-0.0062	(0.0194,0.0244)	(-0.0103,-0.0020)
Buffer Size:				
0.5 RTT	-0.0134	0.0027	(-0.0154,-0.0114)	(-0.0007,0.0060)
1 RTT	0.0016	-0.0008	(-0.0005,0.0036)	(-0.0042,0.0026)
2 RTT	0.0119	-0.0019	(0.0098,0.0139)	(-0.0052,0.0015)
Drop Policy:				
EPD	0.0020	-0.0017	(0.0006,0.0034)	(-0.0041,0.0007)
SD	-0.0020	0.0017	(-0.0034,-0.0006)	(-0.0007,0.0041)

8.3.1 Analysis of Efficiency values: Results and Observations

1. TCP flavor explains 56.75% of the variation and hence is the major factor in deciding efficiency value. Non overlapping confidence intervals for effects of TCP flavors clearly indicate that SACK results in best efficiency followed by NewReno, Reno and Vanilla. However, it should be noticed that difference in performance for different TCP flavors is not very large.

2. Buffer size explains 21.73% of the variation and hence is the next major determinant of efficiency. Confidence intervals for effects of different buffer sizes indicate that efficiency does increase but only slightly as buffer size is increased. However, Vanilla's efficiency increases by about 5% with increase in buffer size from 0.5 RTT to 2 RTT. The corresponding increase in efficiency for other TCP flavors is around 2% or less. This also explains the large interaction between buffer sizes and TCP flavors (explaining 13.42% of the total variation).

3. Drop policy does not cause any significant difference in efficiency values.

Thus, SACK gives best performance in terms of efficiency for MEO links. However, difference in performance for SACK and other TCP flavors is not substantial. For SACK, NewReno and FRR, the increase in efficiency with increasing buffer size is very small. For MEO links, 0.5 RTT is the optimal buffer size for all non-Vanilla TCP flavors with either EPD or SD as drop policy.

8.3.2 Analysis of Fairness values: Results and Observations

As we can see from individual fairness values, there is not much difference between fairness values for different TCP flaovrs, buffer sizes or drop policies. This claim is also supported by the fact that all 9 main effects have very small values, and for 8 of them, their confidence interval encloses 0. Thus, for MEO delays, 0.5 RTT buffer is sufficient for good fairness with any drop policy for all flavors of TCPs.

8.3.3 Simulation Results for GEO links

Table 10 presents the individual efficiency and fairness results for GEO links. Table 11 shows the calculation of 'Total Variation' in GEO results and 'Allocation of Variation' to main effects and first-order interactions. Table 12 shows the 90% confidence intervals for main effects.

Table 10 Simulation Results for GEO Links

Drop Policy	TCP Flavor	Buffer = 0.5 RTT		Buffer = 1 RTT		Buffer = 2 RTT	
		Efficiency	Fairness	Efficiency	Fairness	Efficiency	Fairness
EPD	Vanilla	0.7908	0.9518	0.7924	0.9365	0.8478	0.9496
	Reno	0.8050	0.9581	0.8172	0.9495	0.8736	0.9305
	NewReno	0.8663	0.9613	0.8587	0.9566	0.8455	0.9598
	SACK	0.9021	0.9192	0.9086	0.9514	0.9210	0.9032
SD	Vanilla	0.8080	0.9593	0.8161	0.9542	0.8685	0.9484
	Reno	0.8104	0.9671	0.7806	0.9488	0.8626	0.9398
	NewReno	0.7902	0.9257	0.8325	0.9477	0.8506	0.9464
	SACK	0.9177	0.9670	0.9161	0.9411	0.9207	0.9365

Table 11 Allocation of Variation for GEO Efficiency and Fairness Values

Component	Sum of Squares		%age of Variation	
	Efficiency	Fairness	Efficiency	Fairness
Individual Values	17.3948	21.4938		
Overall Mean	17.3451	21.4884		
Total Variation	0.0497	0.0054	100	100
Main Effects:				
TCP Flavor	0.0344	0.0008	69.16	14.47
Buffer Size	0.0068	0.0006	13.65	11.48
Drop Policy	0.0001	0.0001	0.25	2.31
First-order Interactions:				
TCP Flavor-Buffer Size	0.0037	0.0012	7.54	22.16
TCP Flavor-Drop Policy	0.0025	0.0014	4.96	26.44
Buffer Size-Drop Policy	0.0002	0.0001	0.41	1.45
Standard Error, s_e = 0.0182(For Efficiency), 0.0139(For Fairness)				

Table 12 Main Effects and Their Confidence Intervals for GEO

Factor	Mean Effect		Confidence Interval	
	Efficiency	Fairness	Efficiency	Fairness
TCP Flavor:				
Vanilla	-0.0295	0.0037	(-0.0420,-0.0170)	(-0.0058,0.0133)
Reno	-0.0252	0.0027	(-0.0377,-0.0127)	(-0.0068,0.0123)
NewReno	-0.0095	0.0034	(-0.0220,0.0030)	(-0.0062,0.0129)
SACK	0.0642	-0.0098	(0.0517,0.0768)	(-0.0194,-0.0003)
Buffer Size:				
0.5 RTT	-0.0138	0.0050	(-0.0240,-0.0036)	(-0.0029,0.0128)
1 RTT	-0.0099	0.0020	(-0.0201,0.0004)	(-0.0058,0.0098)
2 RTT	0.0237	-0.0070	(0.0134,0.0339)	(-0.0148,0.0009)
Drop Policy:				
EPD	0.0023	-0.0023	(-0.0049,0.0095)	(-0.0078,0.0033)
SD	-0.0023	0.0023	(-0.0095,0.0049)	(-0.0033,0.0078)

In the following 2 subsections, we present the results of analyzing efficiency and fairness values for GEO links.

8.3.4 Analysis of Efficiency values: Results and Observations

1. TCP flavor explains 69.16% of the variation and hence is the major factor in deciding efficiency value. Confidence intervals for effects of TCP flavors clearly indicate that SACK results in substantially better efficiency than other TCP flavors. Since confidence intervals overlap for NewReno, Reno and Vanilla, one can not be said to be better than other in terms of efficiency.

2. Buffer size explains 13.65% of the variation and interaction between buffer size and TCP flavors explains 7.54% of the variation. Confidence intervals for 0.5 RTT and 1 RTT buffer overlap, thus indicating similar performance. There is a marginal improvement in performance as buffer size is increased to 2 RTT. Vanilla and Reno show substantial efficiency gains as buffer size is increased from 1 RTT to 2 RTT. There is not much improvement for Vanilla and FRR when buffer is increased from 0.5 RTT to 1 RTT. Hence, in this case, 1 RTT buffer does not sufficiently reduce number of packets dropped to cause an increase in efficiency. However, for a buffer of 2 RTT, the reduction in number of dropped packets is enough to improve Vanilla and Reno's performance.

3. Drop policy does not have an impact in terms of efficiency as indicated by negligible allocation of variation to drop policy.

From the observations above, it can be concluded that SACK with 0.5 RTT buffer is the optimal choice for GEO links with either of EPD and SD as switch drop policy.

8.3.5 Analysis of Fairness valu es: Results and Observations

The conclusion here is similar to MEO delays. As we can see from individual fairness values, there is not much difference between fairness values for different TCP flavors, buffer sizes or drop policies. All 9 main effects have very small values, and for 8 of them, their confidence intervals enclose 0. Thus, for GEO delays, 0.5 RTT buffer is sufficient for good fairness with any drop policy for all types of TCPs.

8.4 Overall Analysis

It is interesting to notice how the relative behavior of different TCP flavors change as link delay increases.

As link delay increases, SACK clearly comes out to be superior than NewReno in terms of efficiency. For WAN, SACK and NewReno have similar efficiency values. For MEO, SACK performs a little better than NewReno and for GEO, SACK clearly outperforms NewReno. The reason for this behavior is that NewReno needs N RTTs to recover from N packet losses in a window whereas SACK can recover faster, and start increasing CWND again. This effect becomes more and more pronounced as RTT increases.

SD does not always lead to increase in fairness as compared to EPD. This result can again be attributed to nature of WWW traffic. SD accepts packets of only under-represented VCs after crossing the threshold R. For sufficient buffer size, many of these VCs are under represented in switch buffer because they do not have a lot of data to send. Thus, SD fails to cause significant increase in fairness.

It has been already concluded that for long delay links, end system policies are more important than switch drop policies in terms of efficiency and fairness [GOYAL98]. Results presented in this contribution confirm this conclusion for WWW traffic.

9 Summary

In this contribution we studied the effects of TCP mechanisms, UBR+ drop policies and buffer sizes on the performance of WWW traffic over satellite networks. The following overall conclusions can be made about the efficiency and fairness of WWW TCP traffic over ATM-UBR+ for long delay networks:

Efficiency

1. *End system policies*: SACK generally results in the best efficiency, especially as the delay increases. For lower delay and small buffer sizes, NewReno can perform better than SACK.

2. *Drop policies*: For lower delays (WAN), selective drop improves performance over EPD. As the delay increases, buffer sizes used in our experiments become larger, and selective drop does not have much effect.

3. *Buffer size*: Increasing buffer size increases performance, but the effect of buffer size in much more significant for lower delay.

Fairness

1. *End system policies*: SACK hurts fairness in lower delay (WAN) compared to NewReno. SACK and NewReno have similar fairness for higher delay.

2. *Drop policies*: Drop policies do not have much effect on long delay networks.

3. *Buffer size*: Increasing buffer sizes increases fairness, but for sufficiently large buffers this effect is negligible.

In summary, as delay increases, the marginal gains of end system policies become more important compared to the marginal gains of drop policies and larger buffers.

10 References

[FALL96] K. Fall, S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," Computer Communications Review, July 1996

[FIEL97] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol - HTTP/1.1", RFC 2068, January 1997.

[FLOY95] S. Floyd, "Issues of TCP with SACK," Lawrence Berkeley Labs, Technical report, December 1995

[FLOY98] S. Floyd, T. Henderson, "The NewReno Modification to TCP's Fast Recovery Algorithm," Internet Draft, November 1998, Available from ftp://ftp.ietf.org/internet-drafts/drafts-ietf-tcpimpl-newreno-00.txt

[GOYAL97a] Rohit Goyal, Raj Jain, Shiv Kalyanaraman, Sonia Fahmy, Bobby Vandalore, Xiangrong Cai, Seong-Cheol Kim, Sastri Kota, "Guaranteed Rate for Improving TCP Performance on UBR+ over Terrestrial and Satellite Networks," ATM Forum/97-0424, April 1997, http://www.cis.ohio-state.edu/~jain/atmf/a97-0424.htm

[GOYAL97b] Rohit Goyal, Raj Jain, Shivkumar Kalyanaraman, Sonia Fahmy, Bobby Vandalore, Sastri Kota, "TCP Selective Acknowledgments and UBR Drop Policies to Improve ATM-UBR Performance over Terrestrial and Satellite Networks", Proc. ICCCN97, Las Vegas, September 1997, pp17-27. http://www.cis.ohio-state.edu/~jain/papers/ic3n97.htm

[GOYAL98] Rohit Goyal, Raj Jain, Shivkumar Kalyanaraman, Sonia Fahmy, Bobby Vandalore, "Improving the Performance of TCP over the ATM-UBR Service," *Computer Communications*, Vol 21/10, 1998, http://www.cis.ohio-state.edu/~jain/papers/cc.htm

[HOE96] J.C. Hoe, "Improving the Start-up Behavior of a Congestion Control Scheme for TCP," Proceedings of SIGCOMM96, August 1996.

[KOTA97] Sastri Kota, R. Goyal, Raj Jain, "Satellite ATM Network Architectural Considerations and TCP/IP Performance," Proceedings of the 3rd K-A Band Utilization Conference, 1997, http://www.cis.ohio-state.edu/~jain/papers/kaband.htm

[JAIN91] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Simulation, and Modeling, John Wiley & Sons Inc., 1991.

[LEE96] T. Berners-Lee, R. Fielding, H. Frystyk, "Hypertext Transfer Protocol - HTTP/1.0", RFC 1945, May 1996.

[MAH97] B.A. Mah, "An Empirical Model of HTTP Network Traffic," IEEE INFOCOM'97, April 1997.

[MATH96] M. Mathis, J. Madhavi, S. Floyd, A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, October 1996.

[NS] ns Simulator, Available from http://www-mash.cs.berkeley.edu/ns

[ROMA95] A. Romanow, S. Floyd, "Dymnamics of TCP Traffic over ATM Networks", IEEE JSAC, May 1995.

[SPEC96] SPEC, "An Explanation of the SPECweb96 Benchmark," Available from http://www.specbench.org/osg/web96/webpaper.html

[STEV97] W. Stevens, "TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms," RFC 2001, January 1997.