University of California, Berkeley
Lawrence Berkeley National Laboratory
Distributed System Departament

A Simulation-Based Study of HighSpeed TCP and its Deployment

LBNL-52549

Evandro de Souza

Berkeley - CA - USA
May 2003

Contents

List of Figures

List of Tables

Abstract

The current congestion control mechanism used in TCP has difficulty reaching full utilization on high speed links, particularly on wide-area connections. For example, the packet drop rate needed to fill a Gigabit pipe using the present TCP protocol is below the currently achievable fiber optic error rates. HighSpeed TCP was recently proposed as a modification of TCP's congestion control mechanism to allow it to achieve reasonable performance in high speed wide-area links. In this research, simulation results showing the performance of HighSpeed TCP and the impact of its use on the present implementation of TCP are presented. Network conditions including different degrees of congestion, different levels of loss rate, different degrees of bursty traffic and two distinct router queue management policies were simulated. The performance and fairness of HighSpeed TCP were compared to the existing TCP and solutions for bulk-data transfer using parallel streams.

Keywords: HighSpeed TCP, TCP congestion control, wide area network, bulk data transfer

This publication was elaborated as basis to the thesis to be presented to the Department of Computer Engineering and Industrial Automation, from School of Electrical and Computer Engineering of the State University of Campinas - UNICAMP, in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering.

Disclaimer

Acknowledgments

This work has its foundations in the inspiration, encouragement, help and confidence of a number of people to whom I wish to express my heartfelt gratitude.

First and above all I say thanks to the Almighty God who makes the impossible happens and keeps unchanged his promises.

I would like to express my gratitude to Dr. Eleri Cardozo for his immense confidence on my person and my work.

I would like to express my special thanks to Deborah A. Agarwal for her encouragement, valuable instructions, and support in several areas throughout my period of research at Lawrence Berkeley National Laboratory.

Next, I would extend my sincere gratitude to Sally Floyd for her countless answers, valuable feedback, and inspiration for many of the ideas presented in this work. They were of great help for my study.

I am indebted to Stanley Oliveira for his effort to help writing my thesis. He was a large source of encouragement and support for me.

Futhermore, I would like to mention my gratitude to my family, in special my lovely wife Nilceia. Without her, none of this would have been possible. I extend this thanks to my parents and parents-in-law for their love and encouragement.

Abbreviations

ACK	Acknowledgment
AIMD	Additive Increase Multiplicative Decrease
AQM	Active Queue Management
BDP	Bandwidth Delay Product
ECN	Explicit Congestion Notification
FIFO	First In First Out
FTP	File Transfer Protocol
CWND	Congestion Window
DT	DropTail
DWDM	Dense Wavelength Division Multiplexing
HSTCP	HighSpeed TCP
HTTP	Hyper Text Transfer Protocol
MAX_SSTHRESH	Maximum Slow-Start Threshold
MSS	Maximum Segment Size
MTU	Maximum Transfer Unit
NS-2	Network Simulator Version 2
RED	Random Early Detection
REGTCP	Regular TCP
RFC	Request For Comments
RTO	Retransmit Timer
RTT	Round-Trip Time
SACK	Selective Acknowledge
SSTHERSH	Slow-Start Threshold
SYN	Synchronization Packet
TCP	Transmission Control Protocol
XCP	Explicit Control Protocol
WWW	World Wide Web

Chapter 1

Introduction

One of the major issues in networking today is the increasing demand for more and more bandwidth. Several technologies have emerged and increased the link bandwidth capacity several times. Presently, many technologies are available to link two end points at high speeds, for local area as well as long distance connections.

Today's most import communication media is fiber optics. It was a crucial milestone reached in the network evolution in the last 20 years. Recent development in optical networks produced the DWDM (Dense Wavelength Division Multiplexing) technique. It involves the process of multiplexing many different wavelengths onto a single fiber and opened an extraordinary channel for data transmission. So, the bottleneck point is now shifting from the network link to the end-hosts, when we expect very high-performance communication [27].

With the widespread arrival of applications demanding high bandwidth such as bulk-data transfer, multimedia web streaming, and computational grids for high-performance computing, the networking performance over the wide-area network has become a critical component in the infrastructure [14].

In the Internet, TCP (Transmission Control Protocol) has been widely used as a transport protocol. Many applications such as HTTP (Hyper Text Transfer Protocol) for the Word Wide Web and FTP (File Transfer Protocol) are designed on the basis of TCP. It was developed in the early 1970s, and has been continuously modified since then to adapt to the new applications and link characteristics; and also to improve its capacity.

Recent symptoms have indicated that the congestion control mechanism of TCP has difficult reaching full utilization of optical links, particularly in wide area connections. Network applications are rarely able to take full advantage of these new high-speed networks and they are not utilizing the available bandwidth [28]. The packet drop rate needed to fill a Gigabit pipe using the present TCP protocol is beyond the limit of currently achievable fiber optic error rates, and the congestion control becomes not dynamic [23]. Without expert attention from network engineers, most users are unlikely to achieve even 5 Mbps on a single stream TCP transfers, despite the fact that the underlying network infrastructure can support rates of 100 Mbps or more [30].

These indications have motivated the research around the high speed networks aiming to improve the performance of TCP in situations where there is a high bandwidth delay product. Several proposals have emerged in the literature dealing with some of the issues of this complex problem [17, 39, 55, 54, 35, 36].

On the other hand, keeping the fairness among multiple homogeneous and heterogeneous connections in the network is an essential feature widely accepted in the community [27]. So the new proposed solutions must not interfere too much with the existing solutions, or only interfere when the existing protocols are unable to use the link capacity well.

The proposal of this study is to analyze the deployment of one proposed modification to the TCP congestion control mechanism for use on connections with large congestion windows and low packet drop rate. The HSTCP (HighSpeed TCP) was proposed in 2002 and there exist few studies into the issues of its use.

In this research, the efficacy of HSTCP and the impact of its use on the present implementation of TCP is analyzed in different network conditions. These conditions include different degrees of congestion, different levels of loss rate, different degrees of bursty traffic and two distinct router queue management policies. It is expected that these different network conditions present a broad view of the strengths and weaknesses of HSTCP.

It is also important, for the overall acceptance of this modification, that its performance and fairness be comparable to the existing solutions of bulk-data transfer. So the HSTCP will be compared with parallel streams.

This research is organized as follows. Chapter 2 presents a brief history and the foundations of TCP congestion control. It presents the current problems faced by TCP to achieve high performance, and some of the solutions proposed to overcome these obstacles. Chapter 3 shows the foundation of HSTCP. Chapter 4 shows the proposal for this work and the approach followed. Chapter 5 discusses the methodology used in experiments and metrics for the analyse. The results for the experiments of this study are described in Chapter 6. Chapter 7 presents a discussion about the results found and its meaning. Chapter 8 is dedicated to the conclusion and indications for future work that could follow this research.

Chapter 2

Background

2.1 TCP Overview

2.1.1 Introduction and Congestion Collapse

TCP is a very traditional protocol which was first designed in the early 1970s. Many efforts of research, development and standardization have been extensively devoted to the TCP/IP technology. It is widely used in the current Internet. Many popular Internet services, such as WWW (World Wide Web) and FTP (File Transfer Protocol), use TCP as the de-facto standard transport-layer protocol.

TCP provides reliable segment delivery through a positive acknowledgment mechanism. Each data segment transmitted contains a sequence number indicating the position of the data in the transmission. It is a full duplex protocol, meaning that each TCP connection supports a pair of byte streams, one flowing in each direction. TCP also supports a de-multiplexing mechanism that allows multiple application programs, on any given host, to simultaneously carry on conversations with their peers.

TCP is a sliding window protocol. A sliding window protocol allows the sender to transmit a given number of segments before receiving an ACK (Acknowledgment). When an ACK is received by the sender, the window slides to allow one more segment to be transmitted. It also includes a flow control mechanism for each of the byte streams that allows the receiver to limit how much data the sender can transmit at a given time. Each TCP segment sent (data segments and ACKs) contains a window advertisement. The size of the window advertised by the receiver is the upper bound for the sender's sliding window.

In the first years of the deployment, TCP had only a very rudimentary congestion control mechanism that was not sufficient to prevent congestion in intermediate routers. When too many TCP connections are sending at an inappropriately high rate, the network suffered from "Congestion Collapse" [15]. Congestion collapse is a state in which segments are being injected into the network but very little useful work is being accomplished, most of the data segments or their corresponding ACKs are discarded by one of the intermediate routers in the network before reaching their destination. This causes the sender to retransmit the data, further aggravating the problem. Nagle [45] has a detailed discussion about congestion collapse. TCP's congestion control algorithms attempt to prevent congestion collapse by detecting congestion and reducing the transmission rate accordingly.

Van Jacobson [31] pointed out the importance of the congestion control in the Internet, and proposed some of the TCP algorithms to avoid and control congestion in the network. This work has brought many researchers to become aware of importance of TCP's congestion control. Lakshman [38] examined the performance of TCP/IP over Wide Area Networks. Paxson [48] investigated end-to-end Internet dynamics including the behavior of TCP's congestion control mechanisms, and characterized it. As a result of these efforts, many RFC (Request For Comments) documents regarding TCP were announced to enhance its performance [5, 32].

It is remarkable that TCP is still in use today despite the fact that it was developed 20 years ago. TCP's success is due mainly to its now robust congestion control mechanism. This mechanism causes TCP to reduce its sending rate when congestion is encountered along the network path, as evidenced by dropped packets. Congestion management is imperative in order to allow the network to recover from congestion and operate in a state of low delay and high throughput. It will be presented in the next section two of the most important algorithms in TCP congestion control.

2.1.2 Slow-Start and Congestion Avoidance

Proposed by Van Jacobson [31], the slow-start and congestion avoidance algorithms allow TCP to increase the data transmission rate without overwhelming the network. They use a variable called CWND (Congestion Window). TCP's congestion window is the size of the sliding window used by the sender and cannot exceed the size of the receiver's advertised window. Therefore, TCP cannot inject more than CWND segments of unacknowledged data into the network.

The Slow-Start algorithm is used to increase the amount of unacknowledged data that TCP injects into the network, by gradually increasing the size of the sliding window. Slow-Start is used at the beginning of a TCP connection and in certain instances after detected congestion. The algorithm begins by initializing CWND to one segment. For each ACK received, TCP increases the value of CWND by one segment. For example, after the first ACK arrives, CWND is incremented to two segments and TCP is able to transmit two new data segments. This algorithm provides an exponential increase in the size of the sliding window. Slow-Start continues until either the size of CWND reaches the SSTHRESH (Slow-Start Threshold) or when a segment loss is detected.

The starting value of the CWND is set to that of the MSS (Maximum Segment Size) value. This MSS value is based on the receiver's MSS obtained during the initial TCP handshake, the discovered MTU (Maximum Transfer Unit) path, and the MTU of the sending interface; or, in the absence of other information, 536 bytes.

If the receiver is sending an ACK for every packet, the effect of this algorithm is that the data rate of the sender doubles every RTT (Round-Trip Time) interval. Obviously, this cannot be sustained indefinitely. Either the value of CWND will exceed the receiver's advertised window or the sender's window, or the capacity of the network will be exceeded, causing the loss of packets.

The other limit to the CWND increase during Slow-Start is maintained by the variable SSTHRESH. If the value of CWND increases past the value of SSTHRESH, the TCP flow-control mode is changed from Slow-Start to Congestion Avoidance. Initially the value of SSTHRESH is set to the receiver's maximum window size. However, when congestion is noted, SSTHRESH is set to half the current window size, providing TCP with a memory of the point where the onset of network congestion may be anticipated in the future.

The CWND increase during the Slow-Start phase stops when the congestion window exceeds the receiver's advertised window, when the rate exceeds the remembered value of the onset of congestion as recorded in SSTHRESH, or when it is beyond the network capacity.

When the sending rate is greater than the level which can be sustained by the network, data packets are dropped by the network. TCP can detect packet loss in two ways. First, if a single packet is lost within a sequence of packets, the successful delivery of packets following the lost packets will cause the receiver to generate a duplicate ACK for each successive packet. The reception of these duplicate ACKs is a signal of such a packet loss. Second, if a packet is lost at the end of a sequence of sent packets, there are no following packets to generate duplicate ACKs. In this case, there are no corresponding ACKs for this packet, and the sender's RTO (Retransmit Timer) will expire and the sender will assume packet loss.

In an ACK-based protocol, the sender is responsible for the detection of packet losses. Lost packets are revealed by flaws in the order of acknowledged sequence numbers due to missing ACKs. This ends the Slow-Start phase.

Congestion Avoidance is the phase which follows Slow-Start. In this phase the value of CWND is greater than or equal to SSTHRESH. This algorithm increases CWND at a slower rate than during Slow-Start. For each segment ACKed during Congestion Avoidance, the congestion window is increased by 1/CWND (unless this would make the value of CWND greater than the receiver's advertised window). This adds roughly one segment to the value of CWND every RTT. The Congestion Avoidance algorithm provides a linear increase in the size of TCP's sliding window. This mechanism is used to probe the network for additional capacity in a conservative manner.

The congestion window continues to open in this fashion until packet loss occurs. When a packet loss happens, the resultant duplicate ACKs will trigger the sender to halve the sending rate and continue a linear growth of the congestion window from this new point.

The overall characteristics of the TCP algorithm are an initial relatively fast scan of the network capacity to establish the approximate bounds of maximal efficiency, followed by a cyclic mode of adaptive behavior that reacts quickly to congestion, and then slowly increases the sending rate across the area of maximal transfer efficiency. Packet loss, as signaled by the triggering of the RTO, causes the sender to recommence Slow-Start phase, following a timeout interval. This general behavior is observed in the Figure 2.1.

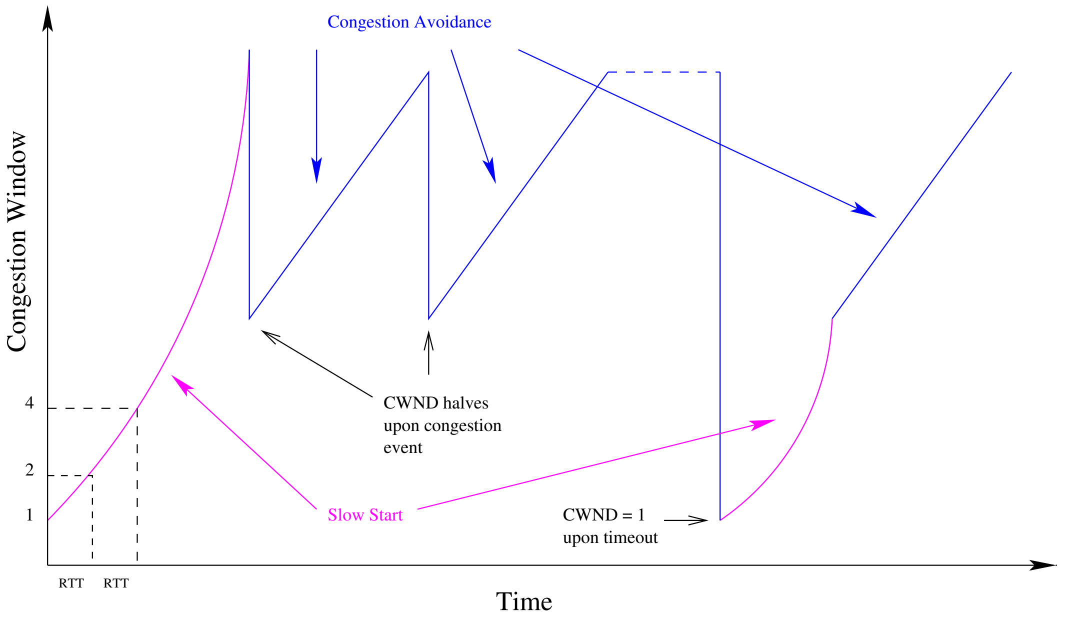


Figure 2.1: TCP Congestion Control

These congestion control algorithms are also known as AIMD (Additive Increase Multiplicative Decrease) and are the basis for the TCP Congestion Control. It increases the congestion window by one packet per window of data acknowledged, and halves the window for every window of data containing a packet drop. In a simple way, TCP congestion control could be expressed in the following equations:

Congestion Avoidance

$$\textbf{ACK} \;:\; CWND \leftarrow CWND + \frac{a}{CWND} \tag{2.1}$$

$$\textbf{DROP} \;:\; CWND \leftarrow CWND - b \times CWND \tag{2.2}$$

Slow-Start

$$\textbf{ACK} \;:\; CWND \leftarrow CWND + c \tag{2.3}$$

The terms CWND, a and c are all defined in units of MSS. The canonical values for a, b and c are: a=1, b=0.5 and c=1.

Some important improvements were incorporated to TCP congestion control, since the early work of Van Jacobson in 1988, that affects the behavior of TCP in high speed connections. Delay Acknowledgment permits the receiver to send cumulative ACKs back to the sender after it receives a predefined number of segments instead of acknowledging every segment. Fast Retransmit defines that when three consecutive duplicate ACKs are received, the sender assumes that the corresponding packet was lost and retransmits it without waiting for the RTO to expire. Fast Recovery is used to avoid entering into Slow-Start after each packet lost. The Window Scale Option addresses the issue of the maximum window size in situations where paths exhibit a high bandwidth delay product. The Timestamp Option allows the sender to calculate a more precise RTT for each received ACK. These improvements are explained in more detail in [56].

One recent important improvement was the introduction of the SACK (Selective Acknowledgment) option. This TCP option alters the acknowledgment behavior of TCP. The SACK option is offered to the remote end during TCP setup as an option in the opening SYN (Synchronization) packet. The SACK option permits selective acknowledgment of permitted data. The default TCP acknowledgment behavior is to acknowledge the highest sequence number of in-order bytes. This default behavior is prone to cause unnecessary retransmission of data, which can exacerbate a congestion condition that may have been the cause of the original packet loss. The SACK option allows the receiver to modify the acknowledgment field to describe noncontinuous blocks of received data, so that the sender can retransmit only what is missing at the receiver's end [42].

The congestion control algorithms for SACK are a conservative extension of Reno's congestion control [13], in that they use the same algorithms to increasing and decreasing the congestion window, and perform better than others when a large number of packets are dropped from a window of data.

Simulations based on performance results of TCP SACK over low and high delay paths were documented in [13]. The results suggested that TCP SACK can significantly improve the network performance when compared with earlier TCP implementations.

2.2 TCP Performance Problems in High Speed Links

This section explains the main problems faced when using TCP to achieve high performance for bulk data transfer in high speed links. The introduction of high-speed network technologies has caused a dramatic change in the achievable performance of TCP based applications. It is well understood that TCP performance depends on network bandwidth, round trip time, and packet loss. As link speed increases, the possibility to achieve a fixed bandwidth-delay product decreases. This problem tends to appear more on high-speed transcontinental links, even though it is now appearing on local area networks [7].

Before exposing these problems, it is necessary to introduce the concept of bandwidth delay product. The capacity of a connection is commonly measured in terms of BDP (Bandwidth Delay Product) [56]. This is mathematically expressed as:

$$Capacity_{(bits)} = bandwidth_{(bits/sec)} \times delay_{(sec)} \tag{2.4}$$

Delay in a network is the two-way latency for information to propagate from the sending node to the receiving one. Bandwidth is the number of bits that can be transmit in a certain period of time. BDP is the product of the above two-performance metrics, i.e. the number of bits (or bytes) the network can hold. You can imagine the network as a pipe, with its length representing the two way network latency and its width representing the bandwidth of the connection, then the BDP is the volume of the pipe. In transport/data link layers, the BDP represents the maximum amount of allowed unacknowledged data outstanding at any moment on the network, keeping the link or pipeline full.

TCP performance depends not upon the transfer rate itself, but rather upon the product of the transfer rate and the round-trip delay. This BDP measures the amount of data that would fill the pipe. The larger the BDP the longer it will take for TCP to use all the capacity available.

Slow-Start

For TCP connections that are able to use large congestion windows of thousands of packets, the current Slow-Start algorithm can result in increasing the congestion window by thousands of segments in a single round-trip time. Such an increase can easily result in thousands of packets being dropped in one round-trip time. This is counter-productive for the TCP flow itself, and it is also hard on the rest of the traffic sharing the congested link.

This drop of a large number of packets can result in unnecessary retransmit timeouts for the TCP connection. The TCP connection could end up in congestion avoidance phase with a very small congestion window, and could take a large number of round-trip times to recover its old congestion window [17].

Frame Size

Today the typical TCP MSS is 1448 bytes, due to the 1500 Byte Ethernet MTU. This size is useful to run at multiple speeds and in a simple switches and hub environment, but it creates problems for applications that need to send a lot of data, such as bulk transfer via FTP, because these applications work better with larger frames. A larger MSS would improve TCP's recovery speed and reduce the packet interrupt rate of the operating system. Other media support a larger MTU: FDDI has a 4392-byte MTU, GigE jumbo-frames have a 9000-byte MTU, IP-over-ATM uses a 9180-byte MTU, and HiPPI uses a 65535-byte MTU. However, these are being phased out in favor of higher speed Ethernet [7, 11].

TCP Buffers

The sender and receiver nodes require buffer space to deal with sending and incoming packets in a connection. This space should be at least the amount of unacknowledged data that TCP must handle in order to keep the pipeline full. TCP performance problems arise when the buffer space is not adequate to accommodate the bandwidth delay product. If the buffers are too small, the TCP congestion window will never fully open up [39].

The receiver's advertised window must also be large enough, since it limits how much the sender can transmit, because the transmitter must not send more data than the advertising window permits. So, the maximum congestion window is proportional to the amount of buffer space that the kernel allocates for each socket. For example, if the RTT is 50 ms and the bandwidth of this connection is 100 Mbits/sec, the TCP buffers should be 625 KBytes. As network throughput capacity has increased in recent years, operating systems have gradually changed the default buffer size from common values of 8 kilobytes to as much as 64 KBytes. However, this is still far too small for today's high speed networks [11], and prevents TCP from using all the bandwidth available.

Congestion Avoidance Algorithm

Presently, TCP implementations can only achieve the large congestion windows necessary to fill a pipe with a high bandwidth delay product, when there is a very low packet loss rate. Random loss leads to a significant throughput deterioration when the product of the loss probability and the square of the bandwidth delay is larger then one [38]. For example, for a standard TCP connection with 1500-byte packets and a 100 ms round-trip time, to achieve a steady-state throughput of 10 Gbps would require an average congestion window of 83,333 segments, and a packet drop rate of at most one congestion event every 5,000,000,000 packets (or equivalently, at most one congestion event every 1h:40m) [18]. This is far beyond what is possible today with the present optical fiber and router technology.

Network Buffer

TCP is bursty in nature. The burstiness of TCP can result in poor performance due to the limited network buffering. Large bursts of data added to the network in a short interval tend to create long queues in the intermediate routers. To fill up a high capacity pipe, TCP needs to use a large maximum window. In most practical cases, the maximum size of the window, which reflects the largest possible size of a traffic burst, is much higher than the queuing capacity of any intermediate router. Once TCP senders overload the router queues, they will start to drop packets. TCP will see these packet drops due to a queuing bottleneck as network congestion. This can result in poor TCP performance like low throughput and unfair sharing. Also, large queues at the routers may introduce additional delays to the TCP flows and increase their RTT [55].

Our research is primarily focused on problems related to the Congestion Avoidance algorithm. However, these problems are closely connect, and may have some impact during the development of this study.

2.3 Proposed Solutions for TCP Performance Problems in HighSpeed Links

After presenting the problems faced by TCP in high speed links, we will overview the research that has been done to overcome them. The results of this research are presented in the following items.

Slow-Start

Limited Slow-Start

Limited Slow-Start is a modification of TCP's Slow-Start algorithm for use with TCP connections with large congestion windows. Limited Slow-Start [17] introduces a parameter called MAX_SSTHRESH (Maximum Slow-Start Threshold). The Slow-Start algorithm is only modified for values of the congestion window greater than MAX_SSTHRESH. The algorithm can be expressed in this way:

For each arriving ACK in Slow-Start:

```
$if(CWND \leq MAX\_SSTHRESH)$
    $CWND = CWND + MSS;$
$else$
    $K = int(CWND/(0.5 * MAX\_SSTHRESH));$
    $CWND = CWND + int(MSS/K);$
```

Thus during Limited Slow-Start the window is increased by 1/K MSS for each arriving ACK, for K = int(CWND/(0.5 * MAX_SSTHRESH)), instead of by 1 MSS as in the standard Slow-Start [1]. When SSTHRESH < CWND, the Slow-Start is exited, and the sender is in the Congestion Avoidance phase.

Frame Size

Large MTU Size

The authors of [43] explain how TCP throughput has an upper bound based on the following parameters:

$$Throughput \approx \frac{1.2 \times MSS}{RTT \times \sqrt{packet\_loss}} \tag{2.5}$$

So, the maximum TCP throughput is directly proportional to the MSS, which is MTU minus TCP/IP headers. If all other things are equal, it is possible to double the throughput by doubling the packet size. Packet loss may also increase with MSS size, but does so at a sub-linear rate, and in any case has an inverse square effect on throughput, i.e. MSS size still dominates throughput [12]. There are proposals to have a Jumbo Frame size, specially for Gigabit Ethernet of 9000 bytes, instead of the present 1500 bytes that is the standard for Ethernet frame size.

Parallel Streams

In order to improve end-to-end performance, parallel TCP streams can be used [39]. This technique is implemented by dividing the data to be transferred into N portions and by transferring each portion with a separate TCP connection. When competing with connections over a congested link, each of the parallel streams will be less likely to be selected for having their packets dropped, and therefore the aggregate amount of potential bandwidth which must go through premature congestion avoidance or slow start is reduced. An application opening N multiple TCP connections is in essence creating a large *virtual MSS* on the aggregate connection that is N times the MSS of a single connection [25].

Experiments have shown that parallel streams can dramatically improve application throughput [39, 58], but this approach is considered to be aggressive and does not provide support for the fair sharing of network bandwidth available to applications [19].

Buffer Management

Automatic Buffer Tuning

The automatic TCP buffer tuning was initially proposed in [54]. It dynamically adjusts the sender's socket buffers to achieve maximum transfer rates on each TCP connection without manual configuration. It is based upon network conditions and system memory availability. As the bandwidth delay product in the Internet can span 4 orders of magnitude, it is not possible to have a single buffer size for all connections in a single machine. If the buffers are tuned, it is possible to avoid waste of kernel memory when the buffer is too large. On the other hand, it is also possible to avoid low throughput results, when the send buffer is small. In this tuning scheme, the sender's flow-control window is tuned.

Dynamic Right Sizing

This proposal is another buffer management technique, proposed in [14]. It proposes that the receiver estimate the bandwidth from the amount of data received in each round-trip time. This estimation is used to dynamically change the receiver's window advertisement, and also to more fairly allocate buffers to connections based in their need for buffers. The growth of the sender's congestion window is limited by currently available bandwidth.

Linux Buffer Tunning

The Linux 2.4 kernel includes TCP buffer tuning algorithms. For applications that have not explicitly set the TCP send and receive buffer sizes, the kernel will attempt to grow the window sizes to match the available bandwidth (up to the receiver's default window). If there is high demand for kernel/network memory, the buffer size may be limited or even shrink. This process is controlled by new kernel variables net.ipv4.tcp_rmem/wmem and the amount of kernel memory available [29].

Congestion Avoidance

XCP

The XCP (eXplicit Control Protocol) [35], generalizes the ECN (Explicit Congestion Notification) proposal. Instead of the one bit congestion indication used by ECN, XCP-enabled routers to inform the senders about the degree of congestion at the bottleneck. Each XCP packet carries a congestion header, which is used to communicate a flow's state to routers and feedback from the routers to the receivers. One field informs the sender's current congestion window, another communicates the sender's current RTT estimate. This information is filled in by the sender and is not modified in transit. The third field is initialized by the sender and receives feedback from routers along the path to directly control the congestion windows of the sources. Like TCP, XCP is a window-based congestion control protocol designed for best effort traffic.

It decouples utilization control from fairness control. To control utilization, this protocol adjusts its aggressiveness according to the spare bandwidth in the network and the feedback delay. To control fairness, the protocol reclaims bandwidth from flows whose rate is above their fair share and reallocates it to other flows. The XCP proposal claims to be stable and efficient regardless of the link capacity, the round-trip time, and the number of sources.

FAST TCP

A congestion control consists of two components, a source algorithm, implemented in TCP, that adapts sending rate to congestion information in its path, and a link algorithm, implemented in routers, that updates and feeds back a measure of congestion to sources that transverse the link. It has been shown that the current algorithms can become unstable as delay increases, and also as network capacity increases. FAST TCP [33] proposes that, to maintain stability, sources should scale down their responses by their individual round trip time and links should scale down their response by their individual capacity. This two actions combined with a dual model for TCP Vegas [41] is claimed to maintain linear stability without having to change de current link algorithm. The authors implemented a FAST TCP kernel with these advances and with some features: it uses both queuing delay and packet loss as signals of congestion; deals with massive losses; reduces burtiness and massive losses using pacing at sender; and converges rapidly to a neighborhood of the equilibrium value and then smoothly home in on the target.

Network Buffer

Paced TCP

Paced TCP is a modified version of TCP that tries to solve the problem of queuing bottlenecks that happen when there is a mismatch between high capacity networks and available storage at the queues of individual network routers [36]. A sender using paced TCP releases packets in multiple, small bursts during a round-trip time, instead of releasing a single large burst of packets, as is the case in ordinary TCP. This approach allows the sender to increase its send rate to the maximum window size without encountering a queuing bottleneck during the Slow-Start.

Chapter 3

HighSpeed TCP Fundamentals

3.1 Description

The HighSpeed TCP for Large Congestion Windows was introduced by Sally Floyd et al. [18] as a modification of TCP's congestion control mechanism for use with TCP connections with large congestion windows. It overcomes Standard TCP's difficulty of achieving a large congestion window in environments with very low packet drop rates. HighSpeed TCP proposes a small modification to TCP's increase and decrease parameters.

In a steady-state environment, with a low packet loss rate p, Standard TCP's average congestion window is roughly 1.2/sqrt(p) segments [19]. This places a serious constraint on the congestion windows that can be achieved by TCP in realistic environments. For example, for a Standard TCP connection with 1500-byte packets and a 100 ms round-trip time, achieving a steady-state throughput of 10 Gbps would require an average congestion window of 83,333 segments, and a packet drop rate of at most one congestion event every 5,000,000,000 packets (or equivalently, at most one congestion event every 1h:40m). If the round-trip time is higher, the time between one congestion event and the next would need to be even greater.

HighSpeed TCP does not modify TCP behavior in environments with mild to heavy congestion, and therefore does not introduce any new dangers of congestion collapse. It is designed to have a different response in environments of very low congestion event rate, and to have the Standard TCP response in environments with packet loss rates of at most 10^{-2}. In environments with low packet loss rates (topically lower than 10^{-3}), it is possible to ignore the more complex response functions that are required to model TCP performance in more congested environments with retransmit timeouts.

The Standard TCP increases its congestion window by one packet per window of data acknowledged, and halves it for every window of data containing a packet drop, following the classical AIMD algorithm. In Congestion Avoidance phase, its behavior is expressed according to equations 2.1 and 2.2 on page 6.

The number of round-trip times between congestion events required for a Standard TCP flow to achieve a high average throughput increases directly with the bandwidth available. For 1500-byte packets and a round-trip time of 0.1 seconds, the figures for congestion window and packet loss rate are [18]:

Throughput (Mbps)	**RTTs Between Losses (secs)**	**Congestion Window (pkts)**	**Loss Rate**
1	5.5	8.3	0.02
10	55.5	83.3	0.0002
100	555.5	833.3	0.000002
1000	5555.5	8333.3	0.00000002
10000	55555.5	83333.3	0.0000000002

Table 3.1: RTTs Between Congestion Events for Standard TCP

3.2 Modified Response Function

The Standard TCP response function, w = 1.2/sqrt(p), gives TCP's average congestion window w as a function of the steady-state packet drop rate p. This Standard TCP response function is a direct consequence of the AIMD mechanisms.

HighSpeed TCP makes use of a different response function and provides a new relation between the average congestion window w and the steady-state packet drop-rate p. For simplicity, this new HighSpeed TCP response function maintains the property that the response function gives a straight line on a log-log scale (as does the response function for Standard TCP, for low to moderate congestion). Both response functions are present in the Figure 3.1.

The HighSpeed TCP response function is specified using three parameters: Low_Window, High_Window, and High_P. Low_Window is used to establish a point of transition and ensure compatibility. The HighSpeed TCP response function uses the same response function as Standard TCP when the current congestion window is at most Low_Window, and uses the HighSpeed TCP response function when the current congestion window is greater than Low_Window. High_Window and High_P are used to specify the upper end of the HighSpeed TCP response function. It is set as the specific packet drop rate High_P, needed in the HighSpeed TCP response function to achieve an average congestion window of High_Window.

The HighSpeed TCP response function can be translated into additive increase and multiplicative decrease parameters. The HighSpeed TCP response function cannot be achieved by TCP with an additive increase of one segment per round-trip time and a multiplicative decrease of halving the current congestion window. It will have to modify both the increase and decrease parameters. That is, HighSpeed TCP has to let the congestion

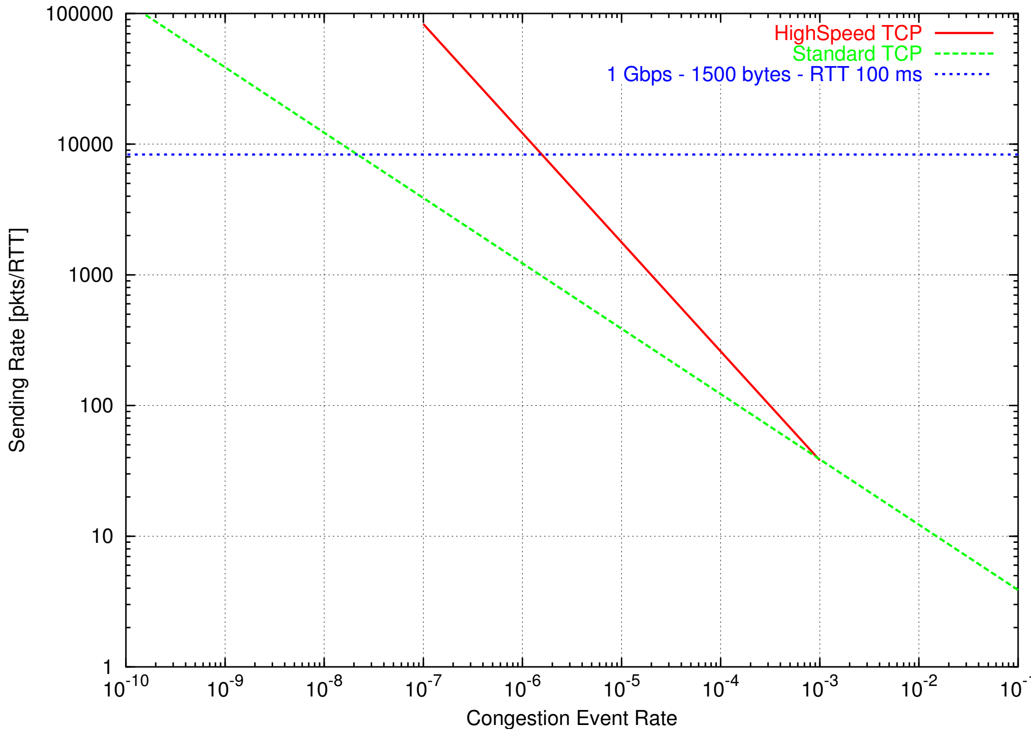


Figure 3.1: HighSpeed TCP Response Function

window increase by a(w) segments per round-trip time in the absence of congestion, and let the congestion window decrease to w(1-b(w)) segments in response to a round-trip time with one or more loss events. In Congestion Avoidance phase, its behavior can be expressed in the following equations:

Congestion Avoidance

$$\textbf{ACK} \quad : \quad CWND \leftarrow CWND + \frac{a(CWND)}{CWND} \tag{3.1}$$

$$\textbf{DROP} \quad : \quad CWND \leftarrow CWND - b(CWND) \times CWND \tag{3.2}$$

We should find the expression for a(w) and b(w) functions based on the three parameters defined for HighSpeed TCP. For w = High_Window, we have specified a loss rate of High_P. For Standard TCP, a(w) = 1 and b(w) = 1/2, regardless of the value of w. HighSpeed TCP uses the same values of a(w) and b(w) for w <= Low_Window. These parameters can be expressed graphically in Figure 3.2.

The HighSpeed TCP maintains the property of having a straight line response function on a log-log scale (as does the response function for Standard TCP, for low to moderate congestion). This results, according to [23], in the following response function, for values of

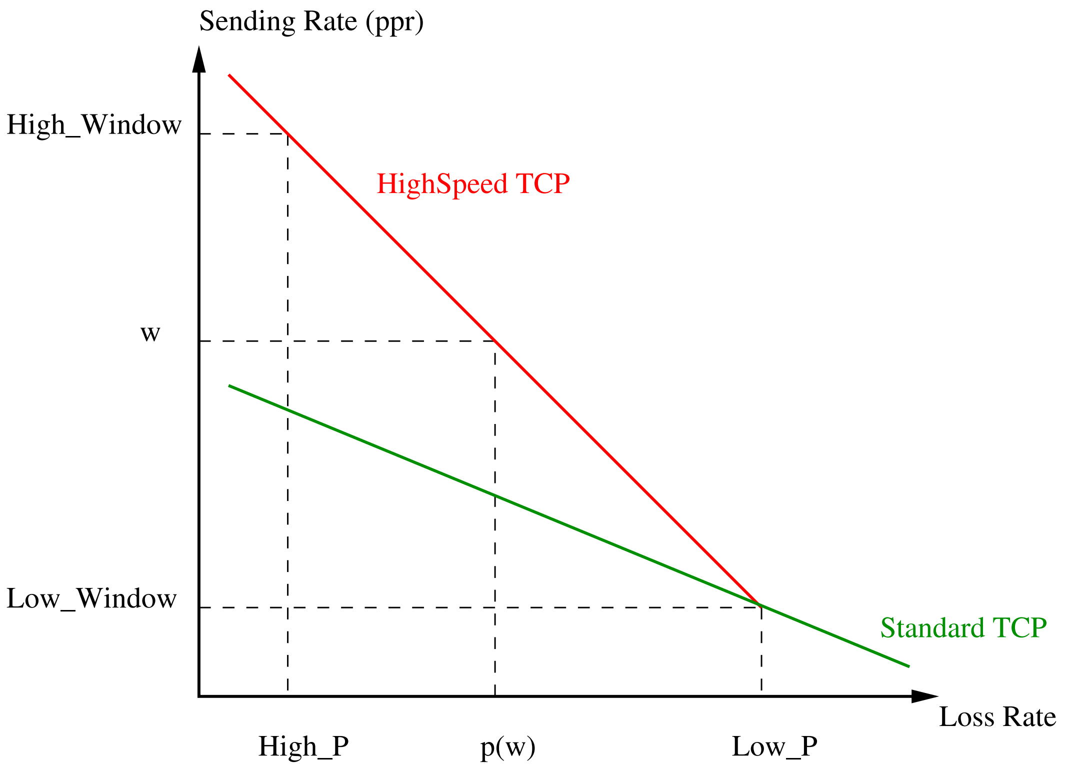


Figure 3.2: HighSpeed TCP Parameters on Log-Log Plot

the average congestion window W greater than Low_Window:

$$W = \left(\frac{p}{Low\_P}\right)^{S} \times Low\_Window \tag{3.3}$$

where Low_P is the packet drop rate corresponding to Low_Window, p is the packet loss rate for this average congestion window W, and S is the following constant:

$$S = \frac{\log High\_Window - \log Low\_Window}{\log High\_P - \log Low\_P} \tag{3.4}$$

From [23], this results in the following relationship between a(w) and b(w):

$$a(w) = \frac{(High\_Window)^2 \times High\_P \times 2 \times b(w)}{2 - b(w)} \tag{3.5}$$

Another parameter High_Decrease is used to specify the decrease parameter b(w) for w = High_Window. The value specified is High_Decrease = 0.1. Given the decrease parameters of b(w) = 1/2 for w = Low_Window, and b(w) = High_Decrease for w = High_Window, it is necessary to specify the values of b(w) for other values of w > Low_Window. From [23], let b(w) vary linearly as the log of w. So, we have :

$$b(w) = \frac{\big(High\_Decrease - 0.5\big) \times \big(\log w - \log Low\_Window\big)}{\log High\_Window - \log Low\_Window} + 0.5 \tag{3.6}$$

$$a(w) = \frac{w^2 \times p(w) \times 2 \times b(w)}{2 - b(w)} \tag{3.7}$$

where p(w) is:

$$p(w) = \Big(\frac{w}{Low\_Window}\Big)^{\frac{1}{S}} \times Low\_P \tag{3.8}$$

3.3 Parameter Selection

To select the Low_Window it is important to choose conservative parameters that provide a backward compatibility with Standard TCP. This requires a response function that is quite close to that of Standard TCP for loss rates of 10^{-1}, 10^{-2}, or 10^{-3}. We set Low_Window to 38 MSS-sized segments, corresponding to a packet drop rate of 10^{-3} for Standard TCP. The decrease parameter b(w) for this point will be 1/2 as it is for Standard TCP. To specify the upper end of the HighSpeed response function, it is necessary to consider the sustained throughput and packet drop rate expected. For example, the average congestion window of 83000 segments is roughly the window needed to sustain 10 Gbps throughput, for a TCP connection with the default packet size of 1500 bytes and round-trip time of 100 ms. For a High_Window set to 83000, it may be specified a High_P of 10^{-7}; i.e., with HighSpeed TCP a packet drop rate of 10^{-7} allows the HighSpeed TCP connection to achieve an average congestion window of 83000 segments. These values set an achievable target for high speed environments, while still allowing acceptable fairness for the HighSpeed TCP response function when competing with Standard TCP in environments with packet drop rates of 10^{-4} or 10^{-5}.

Using the parameters expressed previously, the HighSpeed TCP response function we obtain the figures described in Table 3.2.

Packet Drop Rate P	**Congestion Window W**	**RTTs Between Losses**
10^{-2}	12	8
10^{-3}	38	25
10^{-4}	263	38
10^{-5}	1795	57
10^{-6}	12279	83
10^{-7}	83981	123
10^{-8}	574356	180
10^{-9}	3928088	264
10^{-10}	26864653	388

Table 3.2: TCP Response Function for HighSpeed TCP

3.4 Fairness

It is not difficult to preview the expected relative fairness between HighSpeed TCP and Standard TCP flows. Using both response functions, and defining that Standard TCP has an average congestion window of W_Standard, and HighSpeed TCP a higher average congestion window of W_HighSpeed, the relative fairness will be W_HighSpeed/W_Standard. This is illustrated below. For the parameters chosen for the HighSpeed response function, the relative fairness is described in Table 3.3.

Packet Drop Rate P	**Relative Fairness**
10^{-2}	1.0
10^{-3}	1.0
10^{-4}	2.2
10^{-5}	4.7
10^{-6}	10.2
10^{-7}	22.1
10^{-8}	47.9
10^{-9}	103.5
10^{-10}	223.9

Table 3.3: Relative Fairness between the HighSpeed TCP and Standard TCP Response Functions

The table above can be understood in this way: for a packet drop rate of 10^{-4}, a HighSpeed flow can expect to obtain 2.2 times more throughput than a Standard TCP flow, given the same round-trip times and packet sizes.

Chapter 4

Research Problem Proposal

4.1 Proposal Statement

The general purpose of this work was to study the effectiveness of HighSpeed TCP in high speed long distance links, as a mechanism for bulk data transfer, while maintaining fairness with other types of TCP already in use.

To fulfill this general objective, this study had specific questions to answer as follows:

1. What is the behavior of HighSpeed TCP in situations where Regular TCP underperforms?

2. Is it possible to use HighSpeed TCP together with Regular TCP and maintain an acceptable fairness?

3. What is the effect of the router queuing policy (RED and DT) on the performance of HighSpeed TCP and on the fairness between HighSpeed TCP and Regular TCP?

4. Can HighSpeed TCP be a substitute to other types of bulk data transfer?

4.2 Approach Selection

There were several possible approaches to develop in this investigation. The first approach would be to find an analytical solution for each question. Even though this method provides a precise answer, most times it is very difficult to formulate a broad solution that covers all aspects of an investigation.

The second approach would be to implement a prototype system, deploy it on the Internet in situations that the service was expected to handle, and collect data from its utilization. It would be possible to verify the deployment on the real environment with this method, and it is an excellent goal for wide-area Internet applications. However the costs and difficulty of evaluating systems directly in this way would be prohibitive [24].

The third approach would be to use an emulation of the protocol in a controlled testbed network. Emulation is an interesting tool that offers many of the advantages of direct Internet evaluation, but does not eliminate the problem of obtaining remote computational resources and network access [60]. Emulation allows machines running the actual service on a local-area network to experience delays and bandwidth limitations normally imposed by a wide-area network. This method also would require that the system would be implemented at least in a prototype stage and forces experiments to run no faster than real time.

The next possible approach to this investigation would be to use simulation. This method avoids several costs present in the previous methods and it is indicated as a first analysis method for complex network conditions and behaviors [6]. Using this method it is possible to evaluate network protocols under varying network conditions and investigate unforeseen protocol interactions.

As the present work was one of the first studies of HSTCP, it was reasonable to have simulation as the chosen approach. The other reason for this choice was the availability of good general purpose simulators that were widely accepted in the network research community and had the features required for the development of this study.

4.3 Scope Delimitation

Even though the use of simulation permits the investigation of a protocol in a rich variety of situation, it was not in all network conditions that this would be interesting. We limited the investigation scenarios to selected cases of interest. The main focus was on the behavior of HighSpeed TCP and Regular TCP in situations where both were in steady-state or near to steady-state. The TCP congestion avoidance phase was of particular interest, because it is where the AIMD algorithm works, consequently it is when the HighSpeed TCP algorithm runs. Transient states were also of interest when they made a difference to the behavior or performance in steady-state.

Some consideration was also given to the Slow-Start phase, because of its influence on the overall performance. The algorithm used in Slow-Start has a considerable impact on the congestion avoidance phase. When operating with large congestion windows, it is possible to have thousands of packets dropped from one window and for the recovery to be very slow for the TCP connection.

Long-lived TCP flows are the major interest for high speed and long distance links, when a large amount of data has to be transmitted. So, as a general consideration, most streams used in this work had long duration.

This investigation was developed using a simple topology scenario to avoid more complex interactions and to reduce the number of variables to collect and study.

TCP SACK [13] was used in this work for comparison with HSTCP congestion control, because it had the best existing performance in these situations compared to Tahoe, Reno and Newreno. TCP Vegas was not included in this study due to its reduced performance when competing with other versions of TCP [37], at the time of the development of this study.

The router queuing management was restricted to DT (DropTail) and RED (Random Early Detection) because, the DT is the traditional technique for managing router queue length, and RED is recommended as the best default active queue management mechanism [4]. In the case of RED, ECN [52] was deployed.

Chapter 5

Methodology

In this chapter we present the methodology used in this research. We present the simulation tool used, the network environment used for the tests, the TCP flow types deployed, and how the data was collected. In the end, we describe the metrics and network scenarios used in the evaluation.

5.1 Simulator Selection

The experiments used the NS-2 (Network Simulator - version 2) [50] for the following reasons:

- it is heavily used in the network research community;
- it is a packet simulator that provides a rich network component and protocol library;
- it has good scalability that permits it to be used in different scenarios and different numbers of flows and nodes;
- it is open source software, and can be changed, if necessary;
- the HighSpeed TCP is already implemented in NS-2.

NS-2 has additional features that facilitated this work: the end-nodes and links have parameters that permit adjustment for different scenarios; the routers have implemented DT and RED router queuing management; there are monitoring capabilities for tracking packets at the queue and also within the TCP flows; the simulator includes mathematical support including random number generator and distribution functions. Simulation experiments are written for NS-2 in TCL [59], a rich script language.

5.2 Simulation Environment

This section describes the environment used for the development of this work. All the important conditions and parameters used in the network links, nodes and flows are presented.

5.2.1 Network Topology

The simulation topology chosen was the well-know "dumbbell" [62] with a single bottleneck, as shown in Figure 5.1. This topology provided an excellent platform for studying the effects of the interaction between flows. All traffic in the simulation passed through the bottleneck link and all the end points were connected to it. We defined as "forward" the direction from node N1 to node N2 and as "backward" the direction from node N2 to node N1.

The bottleneck link was the main link. Unless specified, the link bandwidth was 1 Gbits/s, the link delay was 50 ms and the default router queue management type was RED.

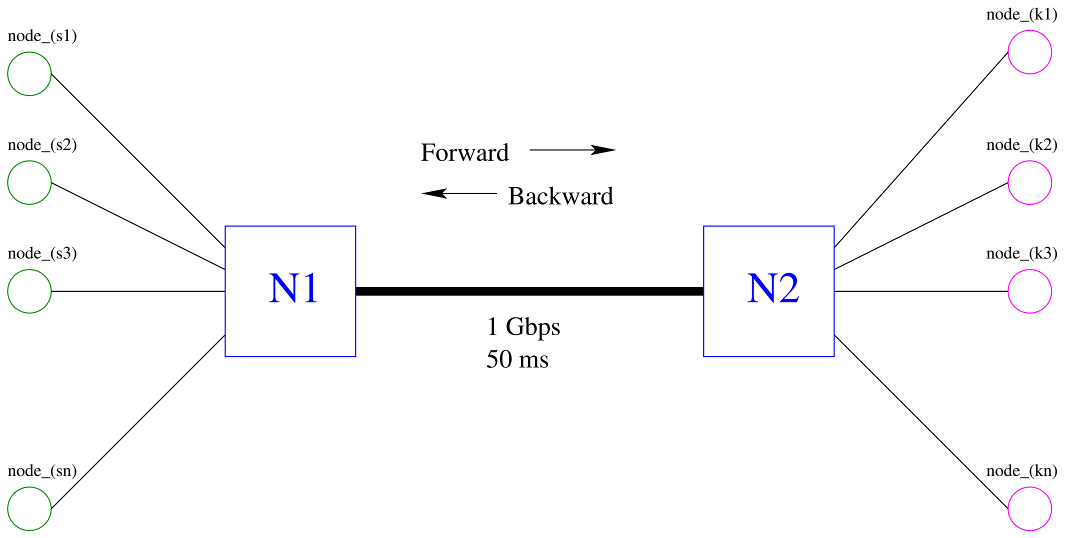


Figure 5.1: Network Topology

We used two types of router queue management, DT and RED. The queue size used was BDP in packets. The default value was 8333 packets, because it matched the BDP for a link of 1 Gbps, with RTT of 100ms and a packet size of 1500 bytes. The other RED parameters are described in Table 5.1.

The Error Model, defined in the NS-2, was used for simulation of link loss. It simulates link-level errors or losses by either marking the packet's error flag or dumping the packet to a drop target. In this work, the Error Model unit was set to "packets", and the

RED Parameters	Value
gentle_	true
adaptive_	1
bottom_	0.0001
thresh_	0
maxthresh_	0
q_weight_	0
drop_tail_	0
setbit_	1
targetdelay_	0.005

Table 5.1: RED Parameters

random variable was set to a "uniform distribution". The error rate is a parameter set based on the level of link-level error desired. This Error Model was only used in the bottleneck link. Unless another value is specified, the error rate in the simulation was set to zero.

End nodes are the end points for the TCP connections. They were linked to the end points of the bottleneck links, N1 and N2. Half the end nodes were linked with N1 and the other half with N2. Each of these links had the bandwidth set to 100,000 Mbps to make sure they were not limiting the TCP flows. They used DT router queue management and a link delay of a few milliseconds (much lower than the bottleneck link). Each node had a slightly different delay to promote behavior diversity among the flows.

5.2.2 TCP Flows Setup

Most flows in the experiments used TCP. These flows shared a set of parameters. They had the ECN bit set to react to ECN-marked packets coming from the router queue management [51, 52]; the packet size was 1500 bytes, this is the size of over half of the byte volume in long distance connections today [44]; the maximum window size was large enough to not impose limits; random times between sends were set to avoid phase effects [21]; unless otherwise specified, the flows used a modified version of the Slow-Start algorithm for large congestion windows to avoid massive packet losses in this phase [17]; and the minimum TCP header was set, with no optional header. The complete TCP parameters are detailed in Table 5.2.

TCP Parameters	Value
ecn_	1
window_	100000
packetSize_	1500
overhead_	0.000008
max_ssthresh_	100

Table 5.2: TCP Parameters

The TCP agent used for the sender and the receiver was SACK1. The TCP/SACK1 implements the BSD Reno TCP transport protocol with Selective Acknowledgment Extensions described in [13]. This TCP flavor implements most of the TCP improvements available nowadays. The HSTCP is also based on this TCP flavor, but with modifications in the congestion control algorithm. FTP was the application used to transmit data through the TCP connections. It simulated bulk data transfer between two nodes.

The HSTCP was implemented in the NS-2 as an option for the congestion window control. Multiple HSTCP flows did not start all at the same time. Rather they started randomly in the first tenth of the total simulation time. The forward direction was used for all HSTCP flows. The values for the HSTCP parameters used are described in Table 5.3.

HSTCP Parameters	**Value**
Low_Window_	31
High_Window_	83000
High_P_	0.0000001
High_Decrease_	0.1

Table 5.3: HSTCP Parameters

For comparison, the HSTCP was run together with Standard TCP implementations, referred to in this work as REGTCP (Regular TCP). These two types of flows used the TCP/SACK1 implementation as explained before. The same basic parameters were used for all the TCP flows, and also these flows start randomly in the first tenth of the total simulation time to avoid phase effects. These flows used only the forward direction to transmit their data.

Since the dominant traffic on the Internet is web-like traffic, this additional flow type was added in the experiments to reduce the regularity of the traffic and obtain more realistic results. The use of this type of flow had the impact of adding competing web-like traffic to the simulations. The PagePool/WebTraf module in the NS-2 was used to provide realistic behavior of web-like traffic. With this module, it is possible to set parameters defining the characteristics of the web servers and web clients (e.g. total number of pages per session, distribution of page size, distribution of inter-arrival pages, distribution of object size). In this work, the average object size was 10, the pool of web servers and web clients was set to 10. Two sets of pools were configured, one in the forward direction (a web server on one side of node N1, and clients on the other side of node N2) and the other pool in the backward direction (web servers on the side of N2 and web clients on the N1 side). The web server and client nodes were linked with the bottleneck nodes through a 100,000 Mbps link and with a variable link delay.

A set of 20 small TCP flows were also used in the simulations, 10 in the forward and 10 in the backward direction. They had a maximum window size of 8 packets. The source and destination of these flows were distributed randomly among the nodes linked to the bottleneck link (the same end nodes as were used for HSTCP and REGTCP flows), so that they could interfere with the HSTCP and REGTCP flows. These flows started at a random time in the first third of the simulation and end at a random time in the last third of the simulation. They represent small short term connections in the Internet.

The previous two types of flows described above, constitute the "background noise" for all the simulations. They were used to avoid having an extremely regular pattern in the experiments and they used less than 1% of the total link capacity available.

The last TCP flow type was used to represent bursty traffic. They were short-lived flows that lasted for about 1.6 seconds. They were in Slow-Start phase during their lifetime and thus had an exponential behavior. The modified version of the Slow-Start algorithm for large congestion windows was not used in this case, instead the standard algorithm was executed. When several bursty flows were used during a simulation, their initial time was randomly distributed over the entire simulation time.

5.2.3 Data Collection Configuration

The NS-2 provides a monitor module to collect data. Two monitors were used in this work, one to monitor the queue in the bottleneck link, tracking the arrival, departure and drop statistics. The other one was used to monitor per-flow statistics. It counted packet and byte arrivals, departures and drops for each flow crossing the bottleneck link.

In addition to these network level statistics, the monitors also provide the ability to collect transport level statistics about TCP as well. It was possible to check how TCP's internal variables were performing. At each simulation tick, information such as current window size, highest sequence number sent, number of ECN responses and slow start threshold were available.

The aggregated data was collected twice, once after half of the simulation time had passed, and other at the end of the simulation. With both sets of data, it was possible to calculate the results for the second half of the simulation. Only the second half was of interest because this research was focused on the steady-state behavior.

Several variables were collected every 0.1 seconds. This was necessary to understand their behavior at intermediate times during the simulation.

The simulation used the NS-2's random number generator. NS-2 implements the minimal standard multiplicative linear congruential generator of Park. This number generator was seeded heuristically.

Each simulation was run ten times, for three hundred seconds. The final result was the median of these simulations. In this work we used a Sun Enterprise Server E4500 with eight CPUs and some Linux boxes. The NS-2 version was NS-2.1b9a or more recent updates available from the NS-2 repository.

5.3 Metrics for Performance Evaluation

The parameters collected in this research are listed and described in Table 5.4.

Type	**Description**	**Unit**
queue size	instantaneous queue size	packets
cong window size	current window size	packets
sequence number	highest acknowledge packet seen by the receiver	—
packet drops	total nr of packets dropped	packets
packet marks	total nr of ECN marked packets	packets
packets sent	total nr of data packets sent by a flow	packets
packet departures	total nr of packets that have departed, not dropped, from the queue	packets
packet arrivals	running total of packets that have arrived in the queue	packets

Table 5.4: Monitor Statistics Collected

A set of metrics was used in this work to evaluate the performance of HSTCP, to measure its impact on other types of TCP traffic, and to verify its behavior in different network conditions. Before we present a formal metric formulation, some definitions should be introduced. They are presented as follows.

Bandwidth B
Description: The number of bits per second that a link is designed to transmit.
Unit: bits/sec

Packet Size pkt
Description: The size of the packets on the network. For the sake of simplicity, the size is fixed.
Unit: bytes

Time Interval T
Description: A period of time.
Unit: seconds

Packets Sent $PS_f(T)$
Description: The number of packets that the flow f transmits during the time interval T.
Unit: packets

Aggregated Packets Sent $APS_p(T)$
Description: The number of packets sent by all flows of the same protocol p during the time interval T, where f is the number of flows belonging to protocol p .
Unit: packets
Expression:

$$APS_p(T) = \sum_{k=1}^{f} PS_k(T)$$

General Packets Sent $GPS(T)$

Description:	The number of packets sent by all protocols belonging to the same experiment during the time interval T, where p is the number of protocols present.
Unit:	packets
Expression:	

$$GPS(T) = \sum_{k=1}^{p} APS_k(T)$$

Link Capacity $C(T)$

Description:	The number of packets that can be transmitted through a link during the time interval T.
Unit:	packets
Expression:	

$$C(T) = \frac{B}{pkt \times 8} \times T$$

Dropped Packets $DP_f(T)$

Description:	The number of packets lost by the flow f during the time interval T.
Unit:	dimensionless

ECN-Marked Packets $ECNP_f(T)$

Description:	The number of ECN marked packets in flow f during the time interval T.
Unit:	dimensionless

Per Flow Congestion Events $FCE_f(T)$

Description:	The sum of the number of dropped plus ECN-marked packets for the flow f during the time interval T.
Unit:	dimensionless
Expression:	

$$FCE_f(T) = DP_f(T) + ECNP_f(T)$$

Congestion Events $CE_p(T)$

Decryption: The sum of the number of dropped plus ECN-marked packets for all flows of the same protocol p during the time interval T, where f is the number of flows of the same protocol.
Unit: dimensionless
Expression:

$$CE_p(T) = \sum_{k=1}^{f} FCE_k(T)$$

General Congestion Events $GCE(T)$
Description: The sum of the aggregated congestion events of all protocols belonging to the same experiment during the time interval T, where p is the number of protocols present.
Unit: dimensionless
Expression:

$$GCE(T) = \sum_{k=1}^{p} CE_k(T)$$

5.3.1 General Metrics

Congestion Event Rate $CER_p(T)$
Description: The ratio between the number of aggregated congestion events and the aggregated packets sent for the same protocol type p during a time interval T.
Unit: dimensionless
Expression:

$$CER_p(T) = \frac{CE_p(T)}{APS_p(T)}$$

General Congestion Event Rate $GCER(T)$
Description: The ratio between the number of general congestion events and the general packet sent for all protocols belonging to the same experiment during the time interval T.
Unit: dimensionless
Expression:

$$GCER(T) = \frac{GCE(T)}{GPS(T)}$$

5.3.2 Link Utilization Metrics

These metrics are used to verify how much of the bandwidth in the bottleneck link is used for a specific flow or for a given protocol. Part of the metrics of this research came from definitions used in [61] and [8].

Per Flow Link Utilization $FLU_f(T)$

Description:	It is the percentage of the bottleneck link utilized by a flow f, during the time interval T in a link with Link Capacity $C(T)$.
Unit:	percentage
Expression:	

$$LU_f(T) = \frac{PS_f(T)}{C(T)} \times 100$$

Link Utilization $LU_p(T)$

Description:	It is the sum of the link utilization of all flows of the same protocol p during a time interval T, where f is the number of flows of a protocol p.
Unit:	percentage
Expression:	

$$LU_p(T) = \sum_{k=1}^{f} FLU_f(T)$$

Bandwidth Stolen $BS_{p1,p2}(T)$

Description:	The link utilization lost by a set of flows belonging to a protocol $p1$ when they compete with N flows from the protocol $p2$, in comparison to the amount of link utilization they obtain when competing with N flows from the same protocol $p1$, during the time interval T.
Unit:	percentage

5.3.3 Fairness Metrics

These metrics are used to verify the impact of the use of a different protocol on an already existent protocol.

Per Flow Relative Fairness $FRF_{f1,f2}$

Description: Applied to assess how many times a flow $f1$ is receiving more average throughput than a flow $f2$, during the time interval T.
Unit: dimensionless
Expression:

$$FRF_{f1,f2} = \frac{FLU_{f1}(T)}{FLU_{f2}(T)}$$

Relative Fairness $RF_{p1,p2}$
Description: Applied to assess how many times a set of flows belonging to a protocol $p1$ is receiving more average throughput than a set of flows belonging to a protocol $p2$, during the time interval T.
Unit: dimensionless
Expression:

$$RF_{p1,p2} = \frac{LU_{p1}(T)}{LU_{p2}(T)}$$

5.4 Descriptions of Scenarios for the Experiments

We used three sets of flows in most of this study. The first set had only HSTCP flows, the second was composed only of REGTCP flows, and the third set contained both REGTCP and HSTCP flows. The first and second flow sets permitted the comparison between REGTCP and HSTCP flows. The third flow set allowed us to observe the interactions between HSTCP and REGTCP flows. The number of flows for each set varied according to each experiment.

These three sets of flows were exposed to different network conditions. The different network conditions permitted us to see the variation of the metrics and produced a picture of the general behavior of HSTCP and its interaction with REGTCP.

In the first network environment there were no other traffic sources and no extra interference beyond that generated by the REGTCP and HSTCP flows. This network environment is referred to as "Ideal Condition" This situation was interesting because it was possible to study how the flow sets performed without external interference. It served as a basis to compare their performance with other network environments.

The second network environment represented the situation where there were systemic losses (or losses not directly related to congestion). It is called "Lossy Link Condition". Some number of packets were randomly dropped from the flows, with a defined drop rate. This allowed us to verify the effects of different link loss levels on the flow sets.

The third network environment explored the reaction of the three flow sets to bursty traffic, so it was called "Bursty Traffic Condition". The bursty traffic was composed of short-lived standard TCP flows running for a few seconds only during the Slow-Start phase, in such a controlled way that it was possible to verify how much they interfered with the behavior of the three flow sets, and consequently with their metrics.

Chapter 6

Results from the Experiments

This section presents the results of several experiments done in order to achieve the objectives proposed in this work. The results are presented here, and the evaluation of their meaning is developed in the next chapter.

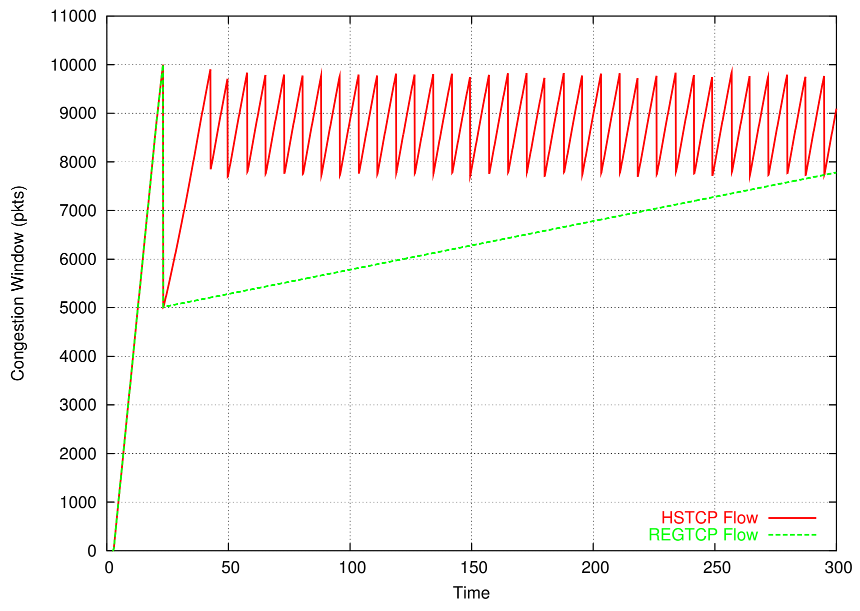
6.1 Isolated Flows

This experiment is intended to observe the basic behavior of isolated REGTCP and HSTCP flows. We used just 1 flow of each type, for 300 seconds and both types of router queue management queue. The experiment ran only one time, without external interference.

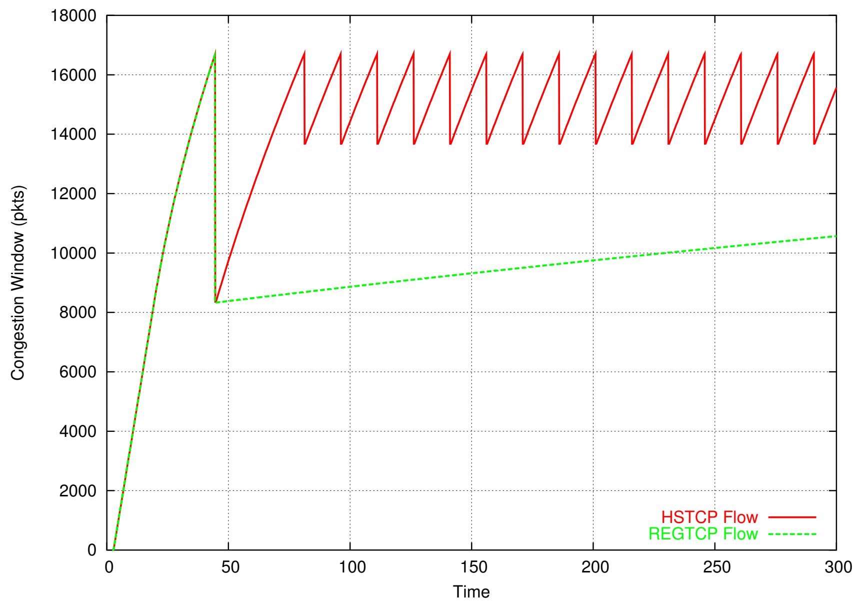
It is possible to observe on Figure 6.1 that the REGTCP flow has a slower growth compared to HSTCP flow. The REGTCP flow reaches the bandwidth limit of 8333 packets around 300 seconds. By its side the HSTCP flow reaches this point before 50 seconds. The second important observation is related to the influence of router queue management type. The congestion window size of both TCP types is higher when DT is used than when RED is used.

6.2 Ideal Condition

This first set of experiments aimed to achieve a baseline in the behavior of REGTCP and HSTCP flows, when there is no external interference, except the background traffic. We used three sets of flows to develop this experiment. The first set contains 1, 2, 6, 10, 20, 30 and 40 HSTCP flows, the second set contains 1, 2, 6, 10, 20, 30 and 40 REGTCP flows and the third one is formed by a mixed of HSTCP and REGTCP flows. It contains 2, 6, 10, 20, 30 and 40 flows, half of each type of flow. Each set of flows was run with RED and DT queuing



(a) RED



(b) DT

Figure 6.1: Evolution of Congestion Window for a Single Flow

policy in the routers. Each simulation ran for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another only by the random number generated each time to begin the simulation. The results show the performance of the selected metrics in this experiment.

Figure 6.2 presents the performance of the aggregated link utilization metric, for the first and second set of flows when RED router queue management is used. DT had similar results.

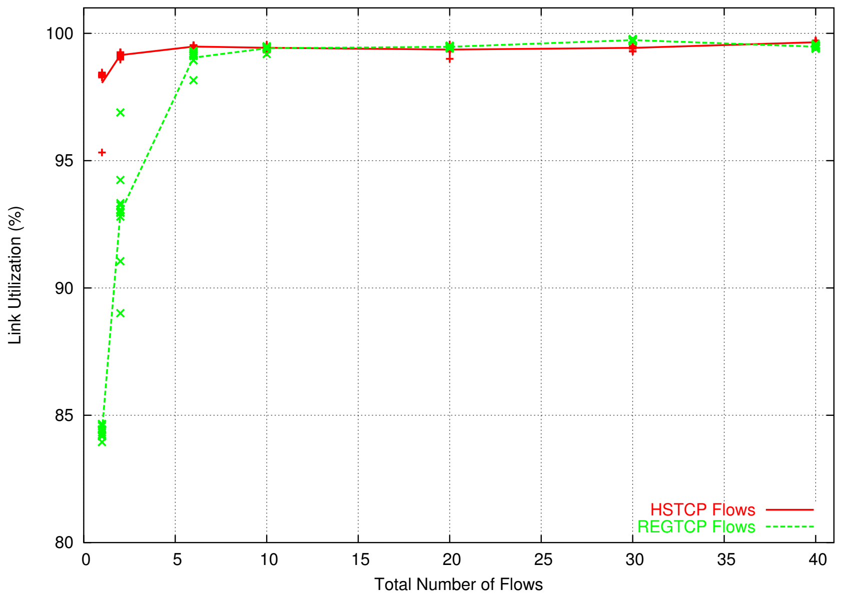


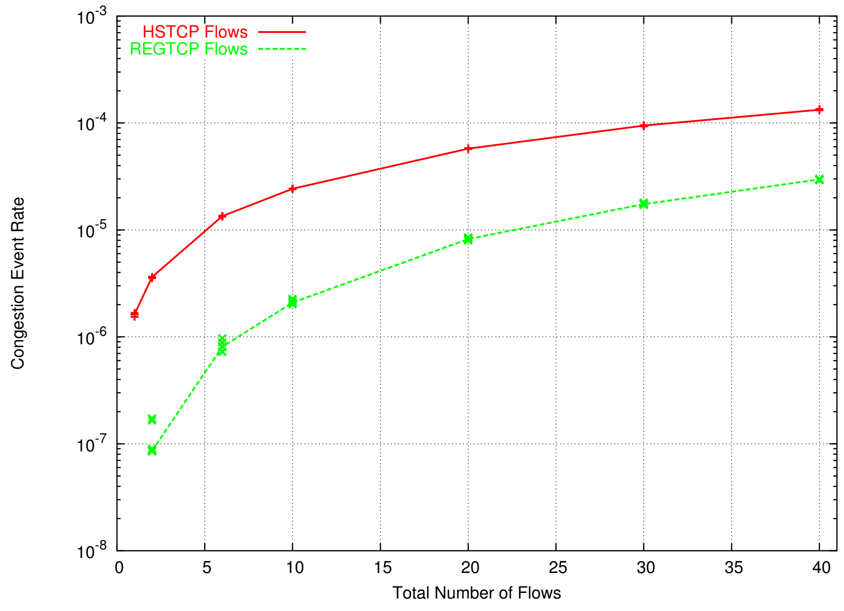
Figure 6.2: Aggregated Link Utilization - Ideal Condition - Homogeneous Flows - RED

This graphic shows that the HSTCP flows can reach full link utilization with a reduced number of flows. Even though the REGTCP flows are using all the bandwidth available, it is clear that they need a higher number of flows to approach to 100% of link utilization.

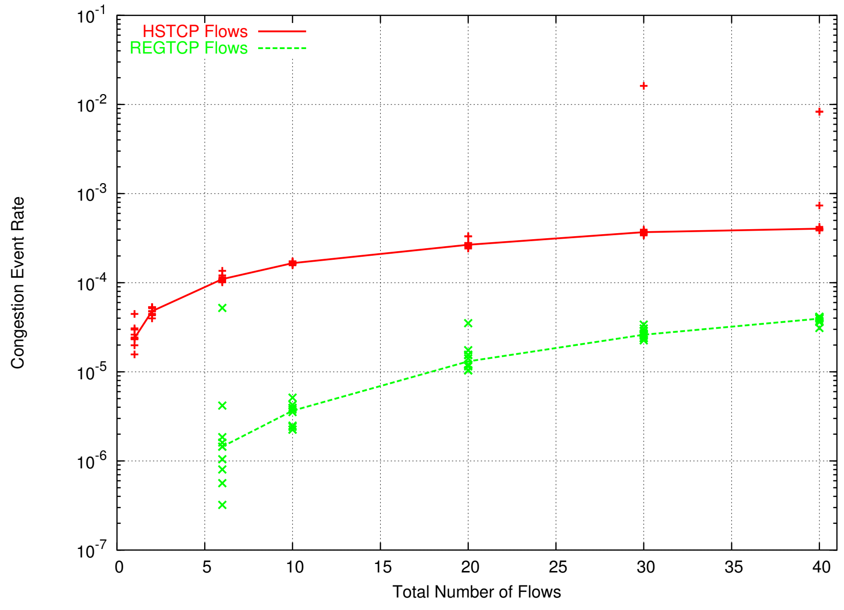
The following two graphics in Figure 6.3 present the aggregated congestion event rate for the first and second set of flows, when RED and DT are deployed, respectively.

These graphics show that there is a clear difference between the congestion event rate resulting from the utilization of each type of flow. The use of HSTCP produces a higher congestion event rate. Another important aspect to observe is that the congestion event rate for HSTCP is never lower that 10^{-6} (RED case for just one HSTCP flow). Finally, the use of DT, as the router queue management policy, generates a higher rate of congestion events than when RED is used.

The link utilization achieved by the third set of flows is presented in the graphs of Figure 6.4. The performance is presented separately for each type of flow. One line is the aggregated result of all HSTCP flows, and the other line is for the aggregated result of REGTCP flows. A third line is the result of all the flows combined. One graph shows



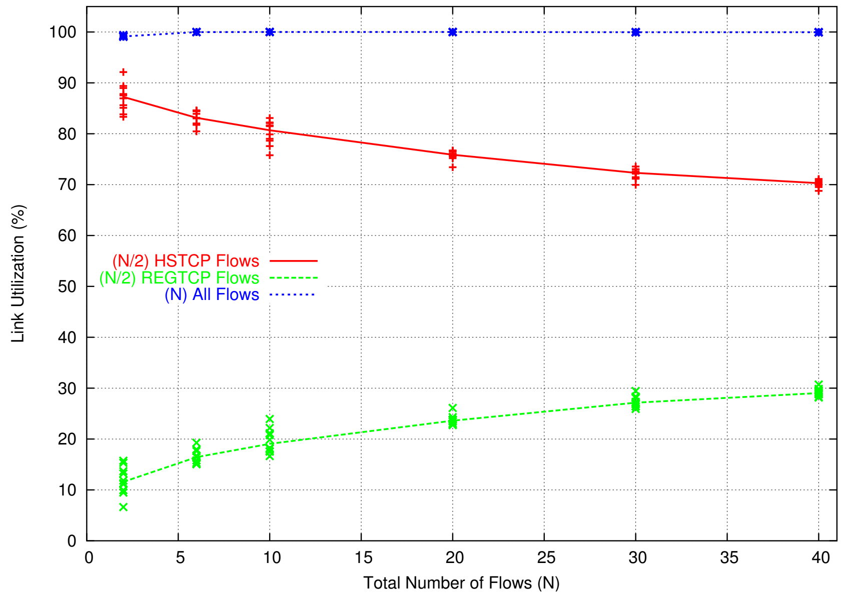
(a) RED



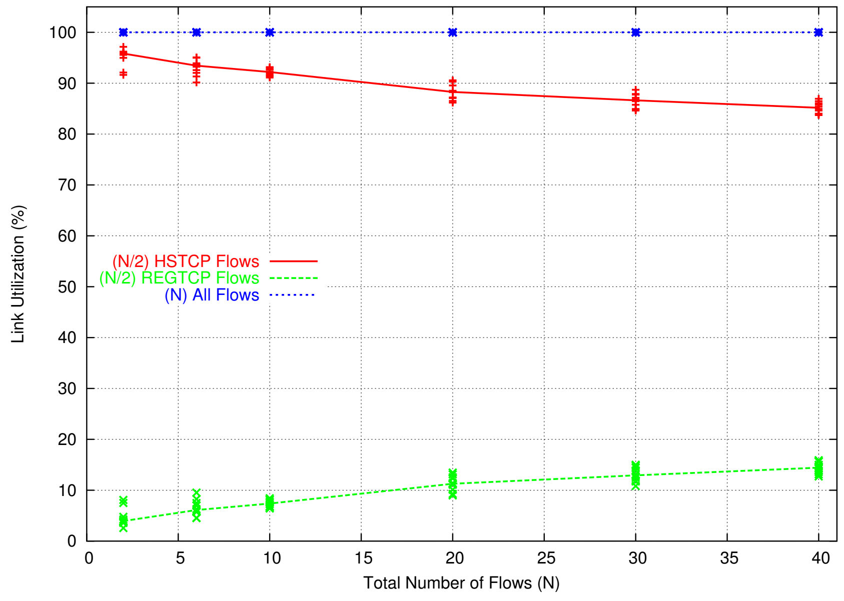
(b) DT

Figure 6.3: Congestion Event Rate - Ideal Condition - Homogeneous Flows

the performance when RED router queue management is used and the other when DT is deployed.



(a) RED



(b) DT

Figure 6.4: Aggregated Link Utilization - Ideal Condition - Heterogeneous Flows

These graphs show that, when HSTCP flows are directly competing with REGTCP flows, the bandwidth share used by HSTCP is higher than the bandwidth used by the

REGTCP flows. This fact is independent of the type of router queue management used. At the other end, the bandwidth share used by HSTCP decreases as the total number of flows increase.

The next Figure 6.5 is the congestion event rate observed for the third set of flows, when both types of flows are deployed together. It shows the results of the RED and DT router queue management.

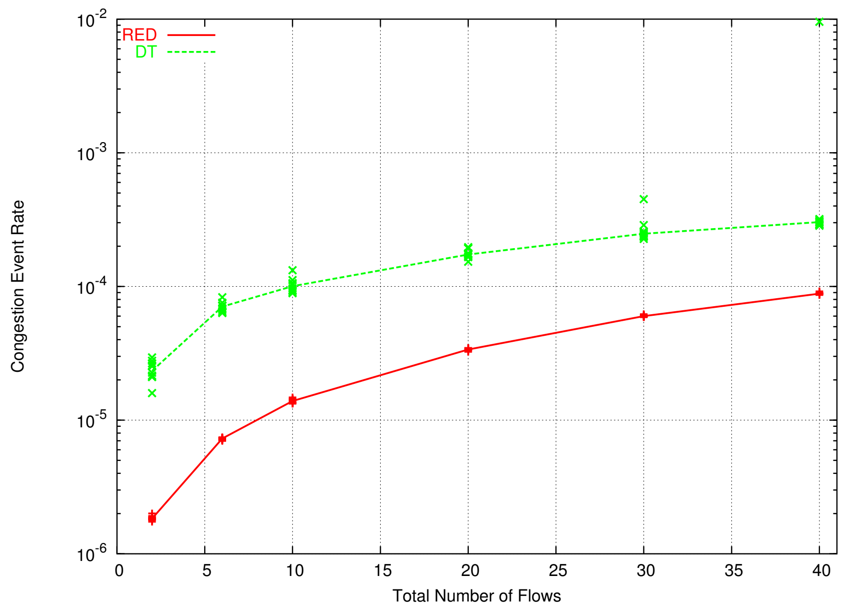


Figure 6.5: Congestion Event Rate - Ideal Condition - Heterogeneous Flows

This graph reveals the evolution of the congestion event rate for this situation, and shows the increase in the congestion event rate as the number of flows increases.

The aggregated relative fairness for the third flow set is depicted in Figure 6.6. It shows the ratio between the amount of bandwidth used by all HSTCP flows and the amount of bandwidth used by all the remaining REGTCP flows.

This graph reveals that the disproportion between the link utilization of both types of flows decreases as the number of flows increases. Another observations that should be pointed out here is the existence of a broad range of fairness values when there is a small number of flows competing for the link. The ratio found in this experiment reached values higher than 35 times. It is also important to observe that when RED is deployed, the aggregated relative fairness is lower than when DT is used.

The last result presented in Figure 6.7 is the amount of aggregated bandwidth stolen from all the REGTCP flows when they are deployed together with HSTCP flows. This result is calculated using the difference between the link utilization achieved by a number of REGTCP flows when they are competing against M other REGTCP flows, and the link

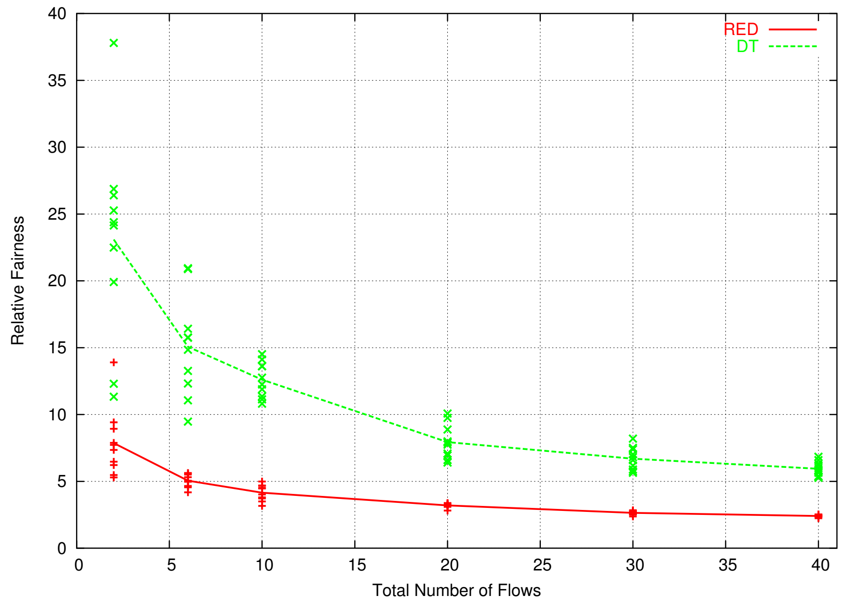


Figure 6.6: Aggregated Relative Fairness - Ideal Condition - Heterogeneous Flows

utilization achieved by the same number of REGTCP flows when they are competing against M other HSTCP flows.

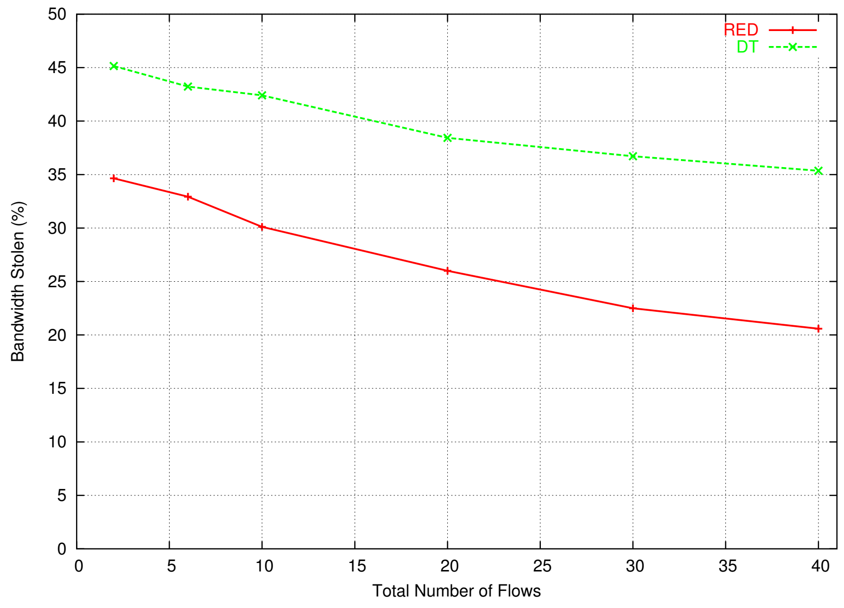


Figure 6.7: Aggregated Bandwidth Stolen - Ideal Condition

This graph shows that the amount of bandwidth stolen decreases as the number of flows increases. This fact highlights that the HSTCP aggressiveness adapts as the traffic condition changes. Other information depicted by this graph is that, even though the amount of bandwidth stolen decreases as the number of flows increase, the distance between the amounts stolen, when RED is used and when DT is used, increases slightly

6.3 Lossy Link Condition

The focus of this set of experiments is to observe the behavior of REGTCP and HSTCP flows when subjected to a systemic losses (losses not due to congestion). We used the simulator error model to simulate losses in the bottleneck link. This loss model was set to drop a packet with a defined average drop rate. The loss rates used were 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, 10^{-2}. We used three sets of flows to develop this experiment. The first set contains 10 HSTCP flows, the second set contains 10 REGTCP flows and the third one is formed by a mix of 5 HSTCP and 5 REGTCP flows. Each set of flow has run with RED and DT queuing policy in the routers. Each simulation ran for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another just by the random number generated to begin the simulation each time. The results show the performance of the selected metrics in this experiment.

Figure 6.8 presents the performance of the aggregated link utilization metric, for the first and second set of flows when RED router queue management is used. DT presents similar results, because, as the link is almost never fully utilized, the router queue management is almost never used.

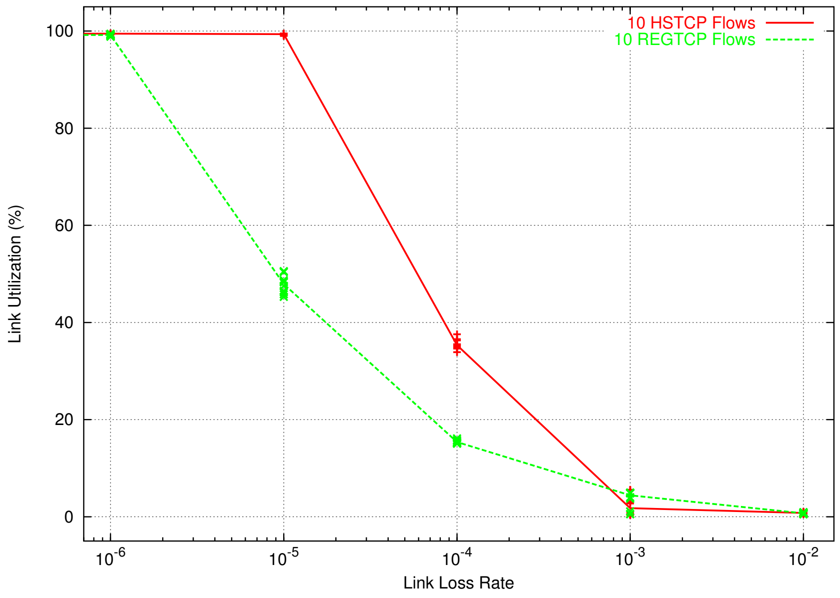


Figure 6.8: Aggregated Link Utilization - Lossy Link Condition - Homogeneous Flows - RED

The set of flows containing only REGTCP flows presents a strong performance loss when the link loss rate increases. This fact indicates that the REGTCP flows are not making reasonable use of the bandwidth available. In contrast, the HSTCP flows presented better performance, and consistently used more bandwidth than the REGTCP flows.

The following two graphics in Figure 6.9 present the aggregated congestion event rate for the first and second set of flows, when RED and DT are deployed, respectively.

There are two points of interest in these graphics. The first one is that the congestion event rate for the set of HSTCP flows is not lower than a limit, 10^{-5} when RED is used and 10^{-4} when DT is deployed. Around this value, there is a *knee* in congestion event rate. The second point to highlight is that the number of congestion events may increase for the set of HSTCP flows near to the bandwidth capacity, as is the case when DT is used.

The big increase in congestion event rates for loss rate higher than 10^{-3} is due to a large number of retransmissions necessary due to the large number of packets lost.

The link utilization achieved by the third set of flows is presented in Figure 6.10. Here the performance is presented separately for each type of flow. One line is the aggregated result of all HSTCP flows, and other line for the aggregated result of REGTCP flows. A third line is the result of all the flows combined.

We see that the difference, between the bandwidth that the HSTCP flows use and the bandwidth that the REGTCP flows are able to use, decreases with the increase in the number of losses. Another important aspect to point out is that, for a link loss rate around 10^{-5}, the link is fully utilized, and below this rate, congestive losses will be dominant.

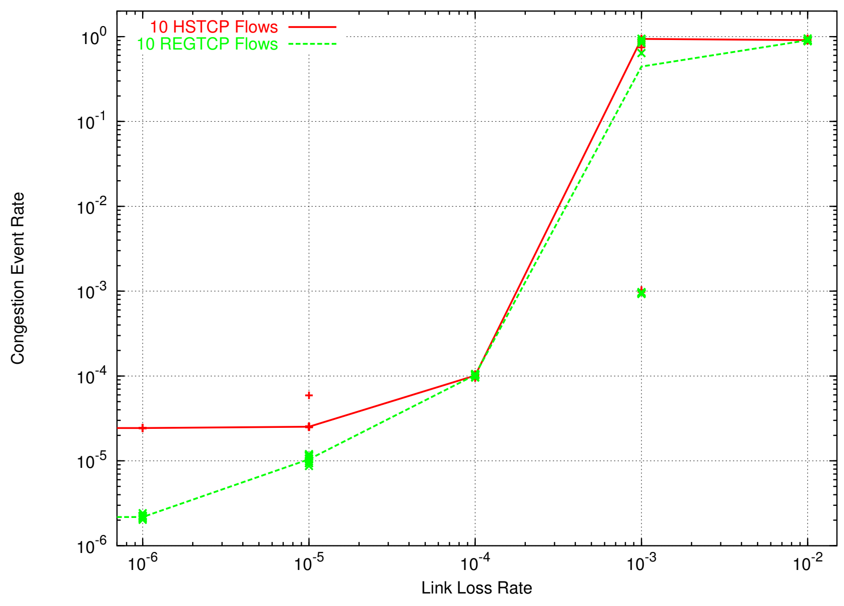
The aggregated relative fairness for the third flow set is depicted in Figure 6.11. It shows the ratio between the amount of bandwidth used by all the HSTCP flows and the amount of bandwidth used for by all the remaining REGTCP flows.

The information about the amount of aggregated bandwidth stolen from all the REGTCP flows when they are deployed together with the HSTCP flows is presented in Figure 6.12. The result is calculated using the difference between the link utilization achieved by a number of REGTCP flows when they are competing against M other REGTCP flows, and the link utilization achieved by the same number of REGTCP flows when they are competing against M other HSTCP flows.

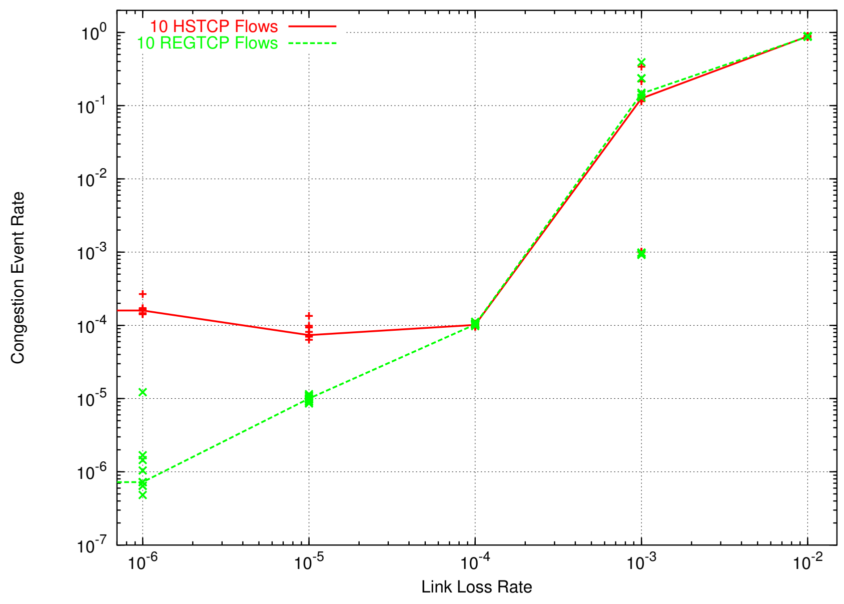
This graphic shows that the REGTCP flows do not lose bandwidth due to deployment of HSTCP when the link loss rate is higher than 10^{-4}. For rates lower than this level, the amount of bandwidth stolen is noticeable.

6.4 Bursty Traffic Condition

The aim of this set of experiments is to understand the behavior and reaction of the REGTCP and the HSTCP flows, when they are submitted to bursty traffic. The bursty



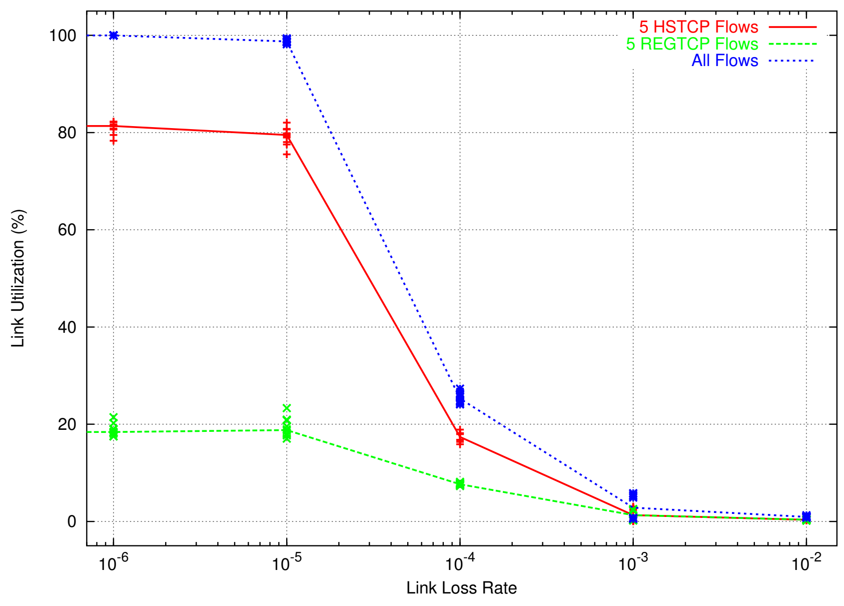
(a) RED



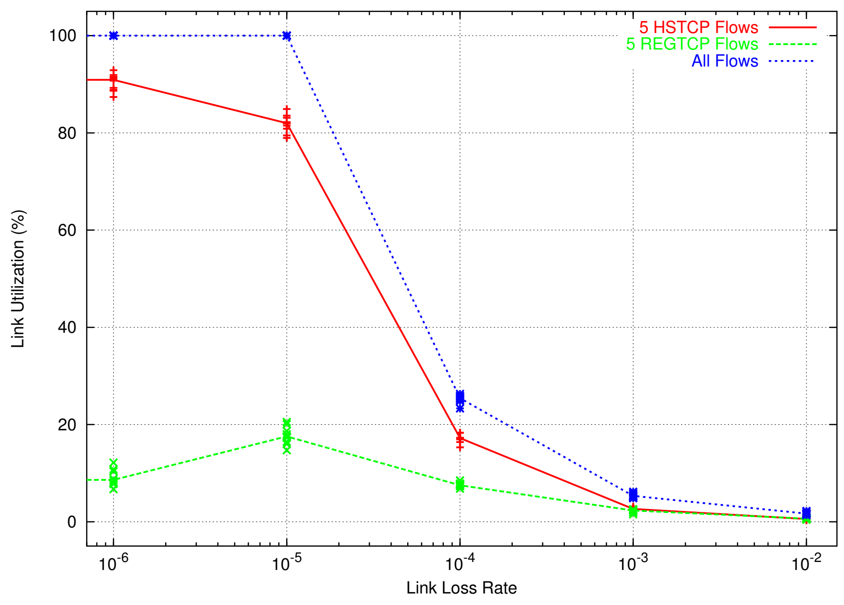
(b) DT

Figure 6.9: Congestion Event Rate - Lossy Link Condition - Homogeneous Flows

traffic is composed of short-lived standard TCP flows running for a few seconds, in such a way that they only run during the Slow-Start phase. As the Slow-Start phase has exponential



(a) RED



(b) DT

Figure 6.10: Aggregated Link Utilization - Lossy Link Condition - Heterogeneous Flows

growth, the bursty flows have a considerable impact in the long-lived flows. We used 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60 and 70 bursty flows, randomly distributed during all the

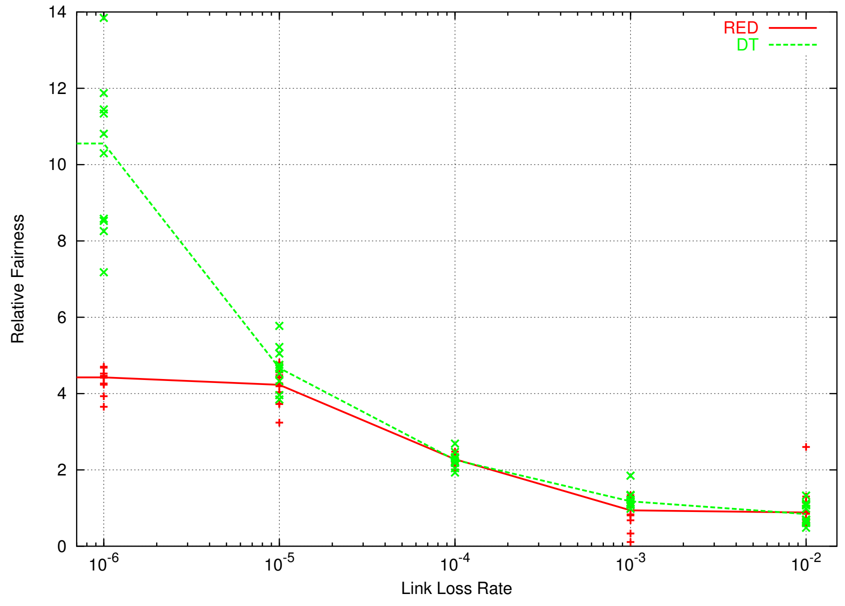


Figure 6.11: Aggregated Relative Fairness - Lossy Link Condition - Heterogeneous Flows

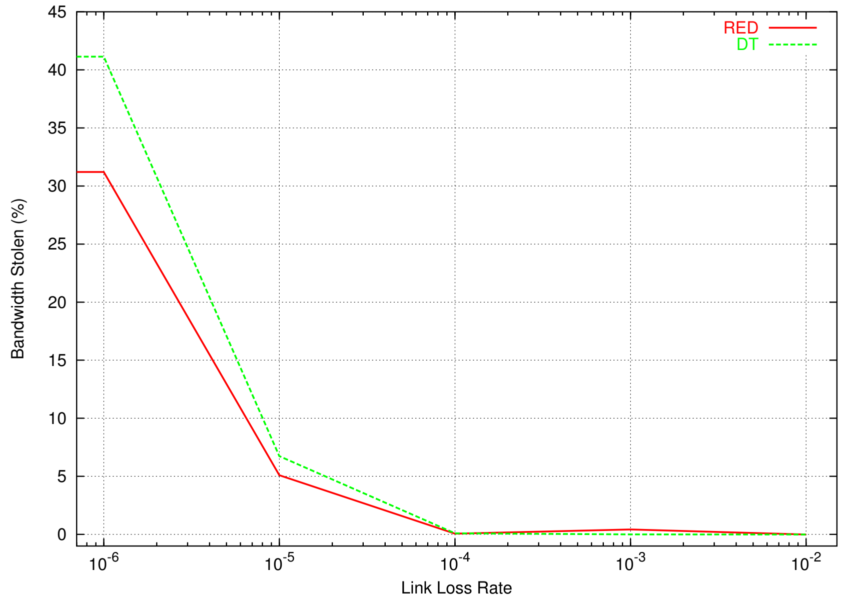


Figure 6.12: Aggregated Bandwidth Stolen - Lossy Link Condition

simulation time.

We used three sets of flows to develop this experiment. The first set contains 10 HSTCP flows, the second set contains 10 REGTCP flows and the third one is formed by a mix of 5 HSTCP and 5 REGTCP flows. Each set of flows was run with RED and DT queuing policy in the routers. Each simulation ran for 300 seconds, and each one was repeated

10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another just by the random number generated to begin the simulation each time. The results show the performance of the selected metrics in this experiment.

The graphs in Figure 6.13 present the performance of the aggregated link utilization metric, for the first and second set of flows when RED and DT router queue management are used, respectively. It also presents the aggregated link utilization of the perturbing flows present when the first set of flows was used and also when the set of REGTCP flows has run.

We observe from Figure 6.13(a) that the set of HSTCP flows decreases their link utilization smoothly and slowly as the number of perturbations increase. On the other hand, the impact on the set of REGTCP flows is higher, and their performance goes down quickly as the number of perturbations increases.

Other information provided by the first graphic is that the amount of bandwidth utilized by a perturbation when competing against the HSTCP flow set is slightly inferior to the bandwidth used when competing against a REGTCP flow set.

The impact of the use of distinct router queuing management is clear when the set of HSTCP flows is submitted to bursty traffic. The link utilization decreases slightly with RED, but it is almost immune to perturbation when the DT router queuing policy is used, as can be seen in Figure 6.13(b).

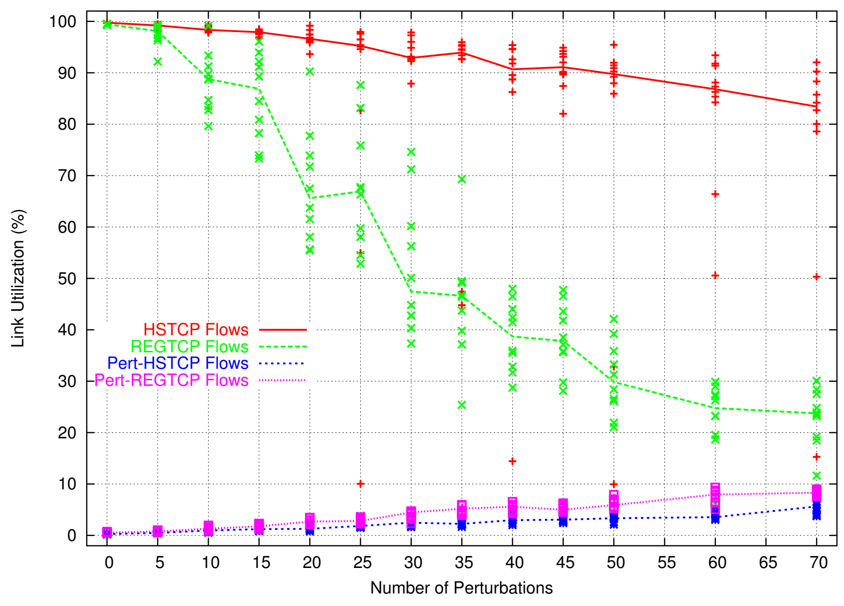
The next two graphics in Figure 6.14 present the aggregated congestion event rate for the first and second set of flows, when RED and DT are deployed.

We observe that the congestion event rate increases continuously as the number of perturbations increases, when RED router queue management is used. This behavior happens with the set of HSTCP flows as well as the set of REGTCP flows. When DT router queue management is deployed, these behaviors change. The set of HSTCP flows presents an almost constant congestion event rate, and the set of REGTCP flows has two levels of congestion event rates, probably caused by the occurrence of global synchronization.

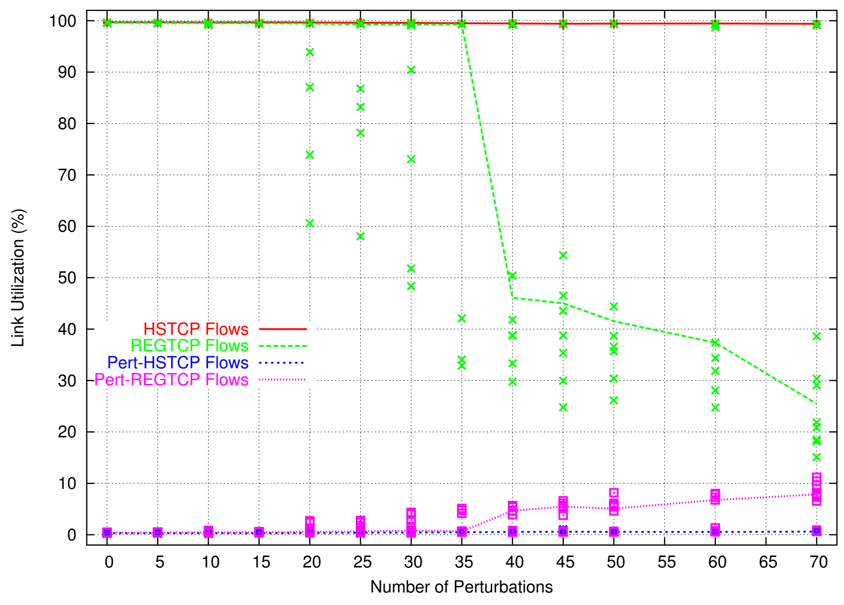
The Figure 6.15 presents the absolute number of congestion events (packets lost plus ECN-marked packets) for the case when RED router queue management is used.

This graphic reveals that the number of congestion events happened, when the set of REGTCP flows was used, was lower than the number from when the set of HSTCP flows was deployed.

The link utilization achieved by the third set of flows is presented in Figure 6.16. It presents the performance separately for each type of flow. One line is the aggregated result of the set of HSTCP flows, and other line is the aggregated result for the set of REGTCP flows. The third line is the result of the all flows combined. The remaining line represents the aggregated link utilization of all the perturbations. One graph shows the performance



(a) RED

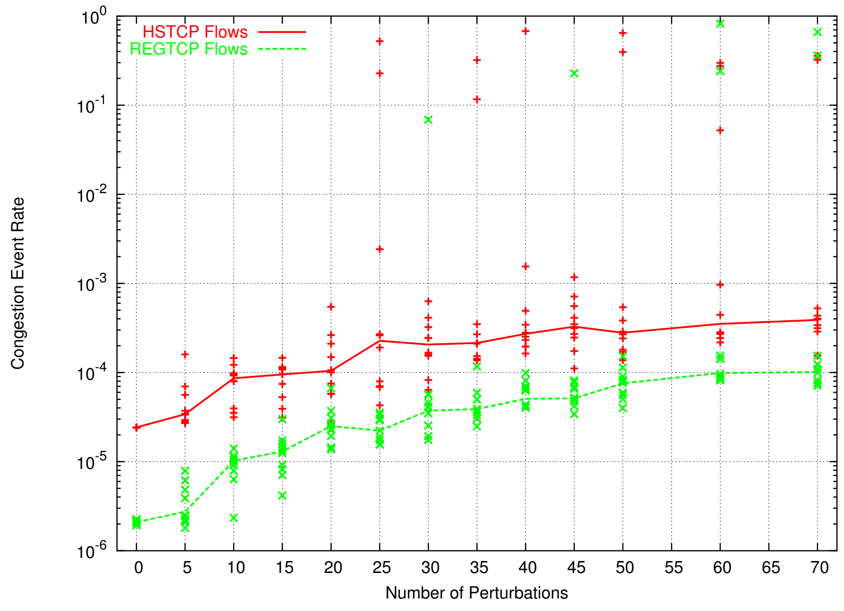


(b) DT

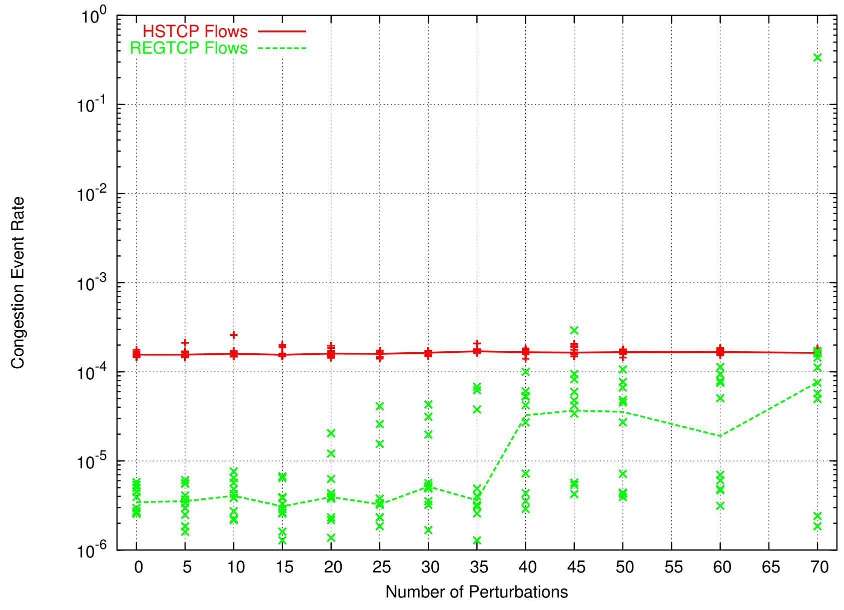
Figure 6.13: Aggregated Link Utilization - Bursty Traffic Condition - Homogeneous Flows

when RED router queue management is used and the other when DT is deployed.

The important information provided by these graphics is the poor and almost con-



(a) RED



(b) DT

Figure 6.14: Congestion Event Rate - Bursty Traffic Condition - Homogeneous Flows

stant performance of the set of REGTCP flows. They have a low link utilization, however, they do not change this performance much when more perturbations are used.

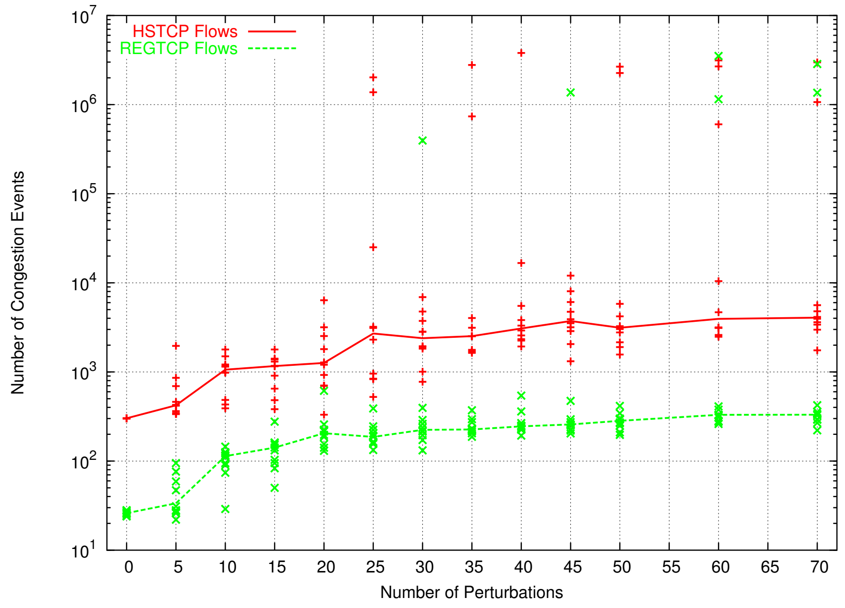


Figure 6.15: Congestion Events - Bursty Traffic Condition - Homogeneous Flows - RED

The aggregated relative fairness for the third flow set is depicted in Figure 6.17. It shows the ratio between the amount of bandwidth used by all the HSTCP flows and the amount of bandwidth used by all the remaining REGTCP flows.

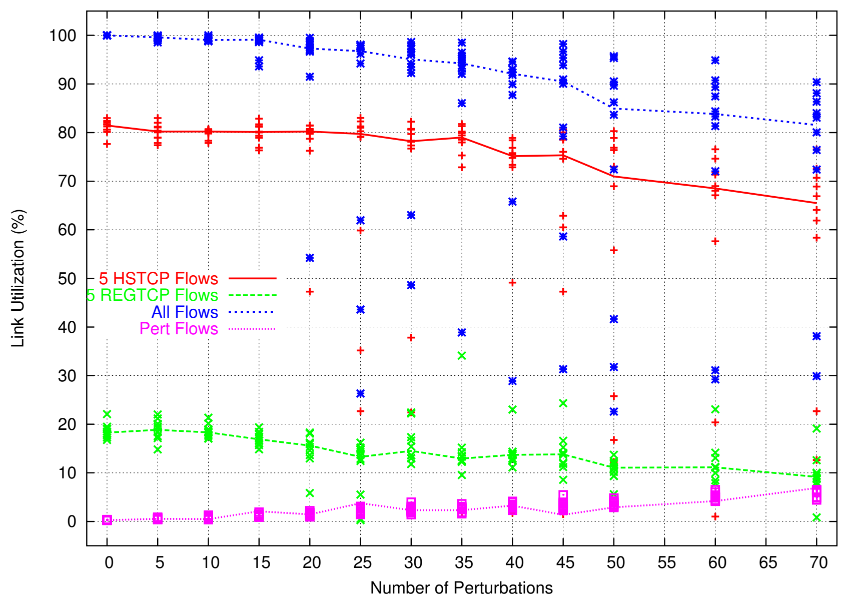
The relative fairness is almost constant using RED, as well as when DT was used. The HSTCP flows get between 10 and 15 times more bandwidth share than the REGTCP flows, for DT, and they get around 5 times, when RED is used. The level of relative fairness in RED grows slightly as the number of perturbations increases.

The aggregated bandwidth stolen from all the REGTCP flows when they are deployed together with the HSTCP flows is presented in Figure 6.18.

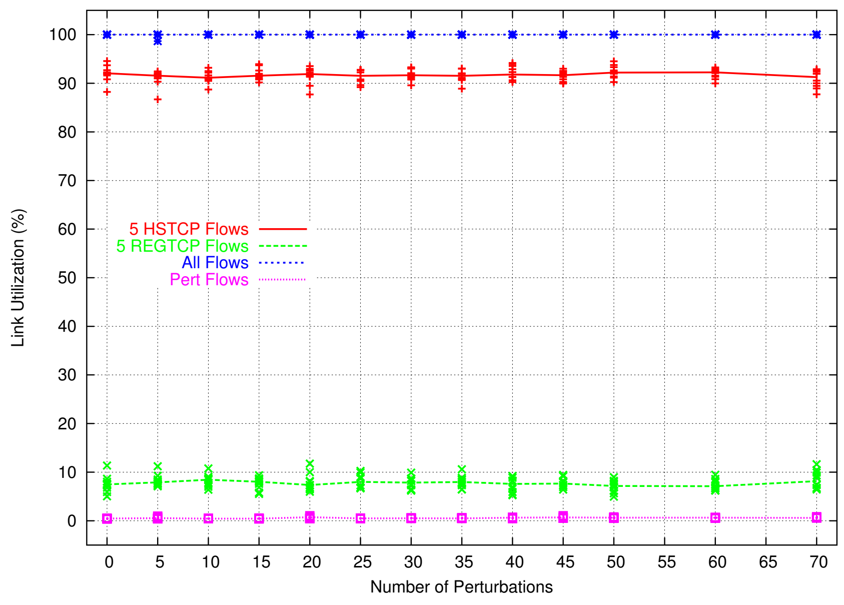
The figure highlights that the amount of bandwidth stolen by the HSTCP flows from the REGTCP flows decreases as the number of perturbations increases, independent of the type of router queue management used. But, the amount of bandwidth stolen is higher for DT router queue management.

6.5 Competition among Heterogeneous Flows

In this set of experiments, we wanted to verify the behavior of the HSTCP and the REGTCP flows when an asymmetric number of flows is deployed. To fulfill this objective, we ran 1 HSTCP flow against varying number of REGTCP flows. We used 1, 3, 5, 7, 11, 15 and 19 REGTCP flows. These flows were run with RED and DT queuing policy in the



(a) RED



(b) DT

Figure 6.16: Aggregated Link Utilization - Bursty Traffic Condition - Heterogeneous Flows

routers. Each simulation runs for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs

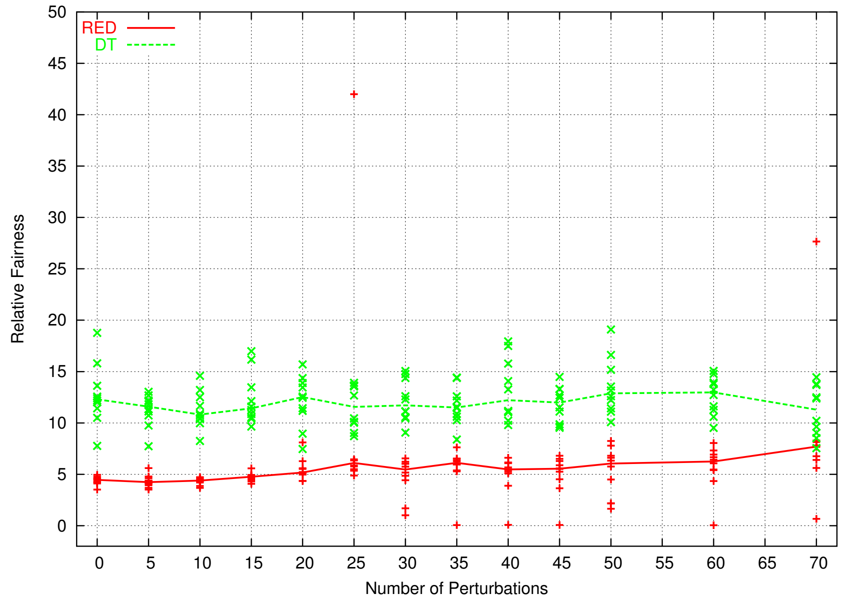


Figure 6.17: Aggregated Relative Fairness - Bursty Traffic Condition - Heterogeneous Flows

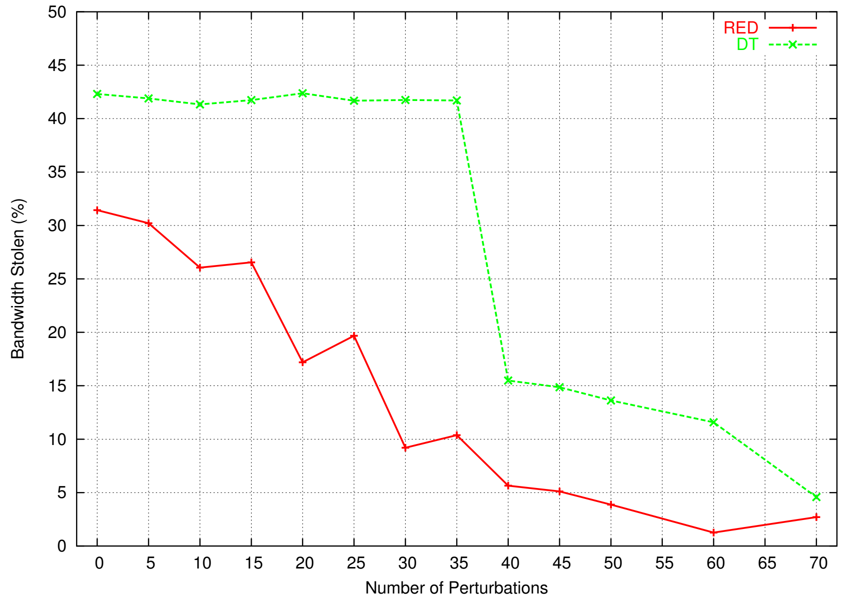
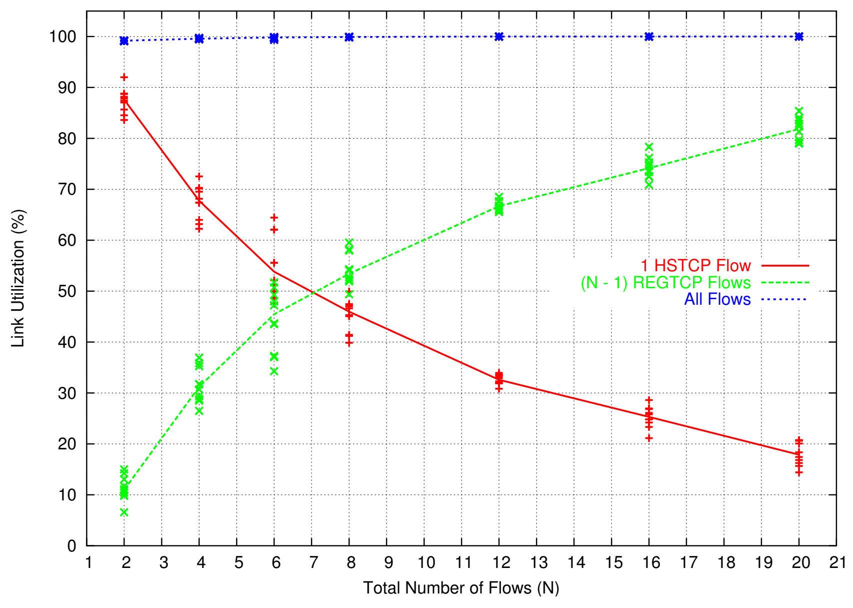


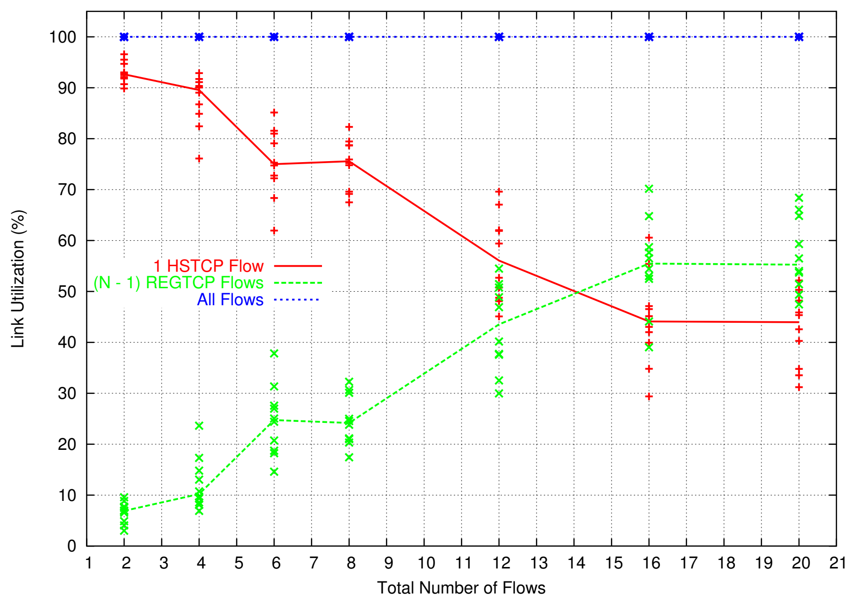
Figure 6.18: Aggregated Bandwidth Stolen - Bursty Traffic Condition

from another just by the random number generated to begin each simulation each time. The results show the performance of the selected metrics in this experiment.

The graphs in Figure 6.19 present the performance of the aggregated link utilization metric when RED and DT router queue management are used, respectively. It also presents the aggregated link utilization of all the flows together.



(a) RED



(b) DT

Figure 6.19: Aggregated Link Utilization - Competition Among Heterogeneous Flows

Some important information is present in these graphics. The first one is that the HSTCP flow adapts itself with the amount of REGTCP flows used, and it avoids the link to become idle. The second piece information is that there is a certain number of REGTCP flows that has equivalent performance to 1 HSTCP flow, but this number is dependent on the type of router queue management used. This equivalence happens at the crosspoint of the HSTCP line and the REGTCP line.

The aggregate relative fairness is depicted in Figure 6.20. It shows the ratio between the amount of bandwidth used by the HSTCP flow and the amount of bandwidth used by all the other REGTCP flows.

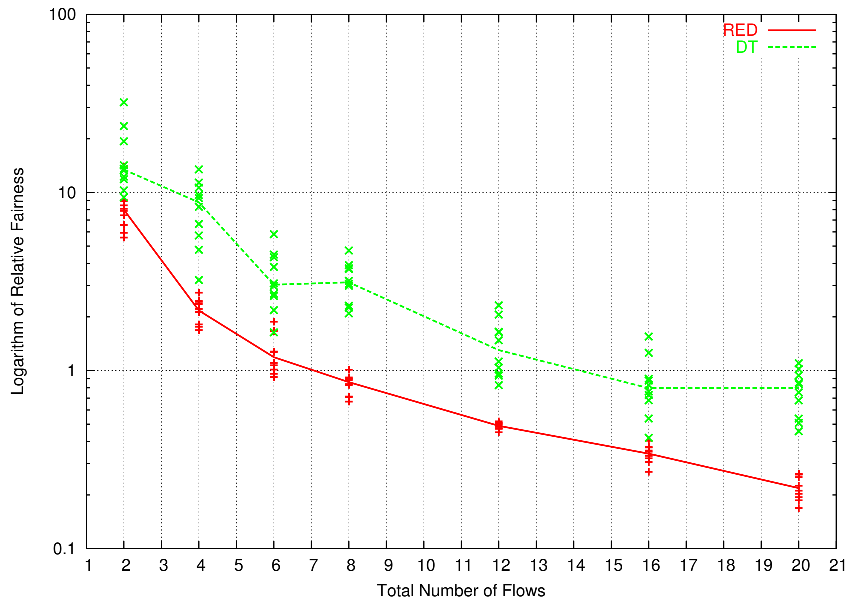


Figure 6.20: Aggregated Relative Fairness - Competition Among Heterogeneous Flows

6.6 Constant Link Loss of 10^{-5}

This set of experiments repeats the experiment of Ideal Condition, except that it introduces a constant link loss rate of 10^{-5}. The purpose of this change is to investigate the behavior of the HSTCP and the REGTCP flows with systemic loss, but not as done in the experiment of Lossy Link Condition, where each set of flows had only 10 flows. Instead, it used a variable number of flows for each set of flows.

We used three sets of flows to develop this experiment. The first set contains 1, 2, 6, 10, 20, 30 and 40 HSTCP flows, the second set contains 1, 2, 6, 10, 20, 30 and 40 REGTCP flows and the third one is formed by a mix of HSTCP and REGTCP flows. It contains 2, 6, 10, 20, 30 and 40 flows, half of each type of flow. Each set of flows was run with RED and DT queuing policy in the routers. Each simulation ran for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another just by the random number generated to begin each simulation each time. The results show the performance of the selected metrics in this experiment.

Figure 6.21 presents the performance of the aggregated link utilization metric for the first and second set of flows when RED router queue management is used. DT presents similar results.

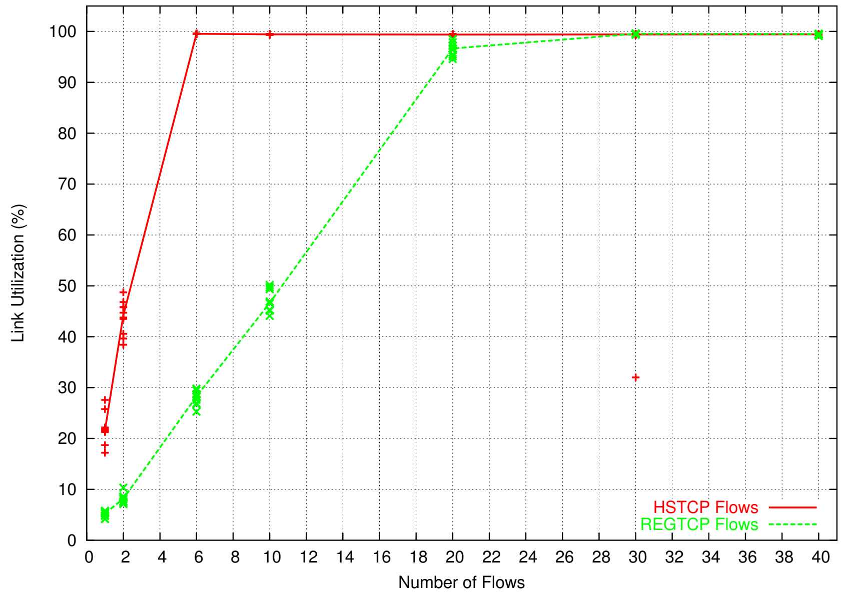


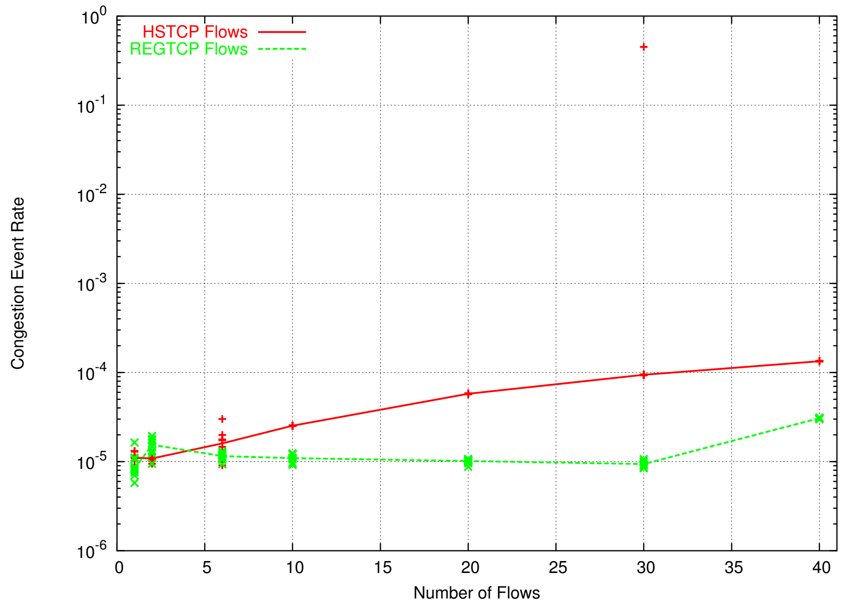
Figure 6.21: Aggregated Link Utilization - Constant Link Loss Rate of 10^{-5} - Homogeneous Flows - RED

The information provides in this graph is that, when HSTCP flows are deployed in this network condition, there is need of 6 flows to reach full link utilization. But, when REGTCP flows are used, this number increases to 20 or more.

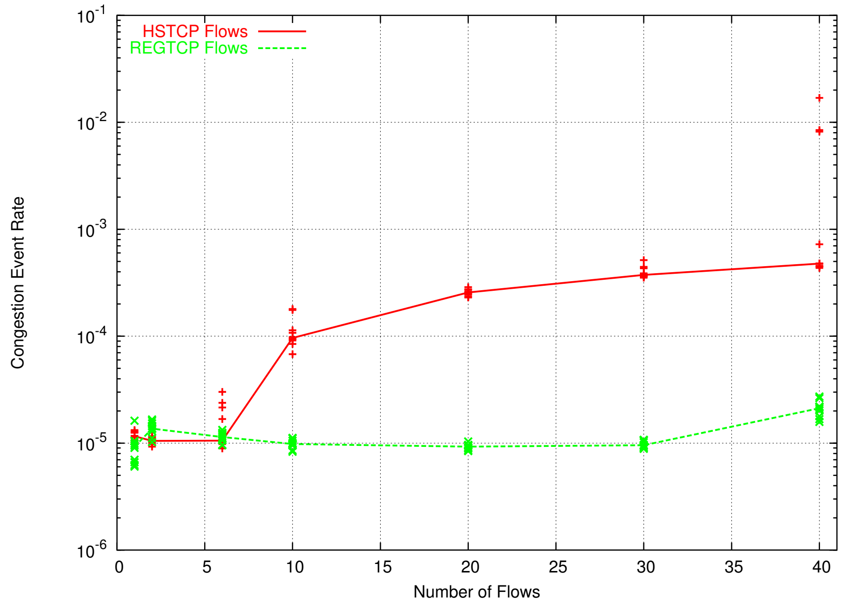
The two graphs in Figure 6.22 present the aggregated congestion event rate for the first and second set of flows, when RED and DT are deployed, respectively.

The congestion event rate for both sets of flows presents a change in its behavior. This aspect can be seen when there are more than 30 REGTCP flows, and when there is more than 6 HSTCP flows. This so called *knee* represents the point when full link utilization is reached.

The link utilization achieved by the third set of flows is presented in Figure 6.23. It presents the performance separately for each type of flow. One line is the aggregated result of all the HSTCP flows, and other line is for the aggregated result of the REGTCP flows. A third line is the result of all the flows combined. One graph shows the performance when RED router queue management is used and the other when DT is deployed.



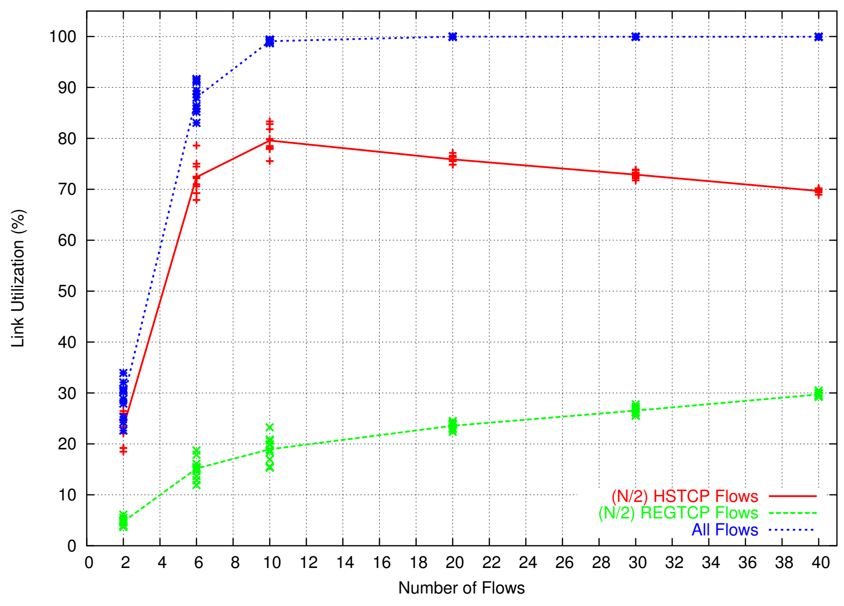
(a) RED



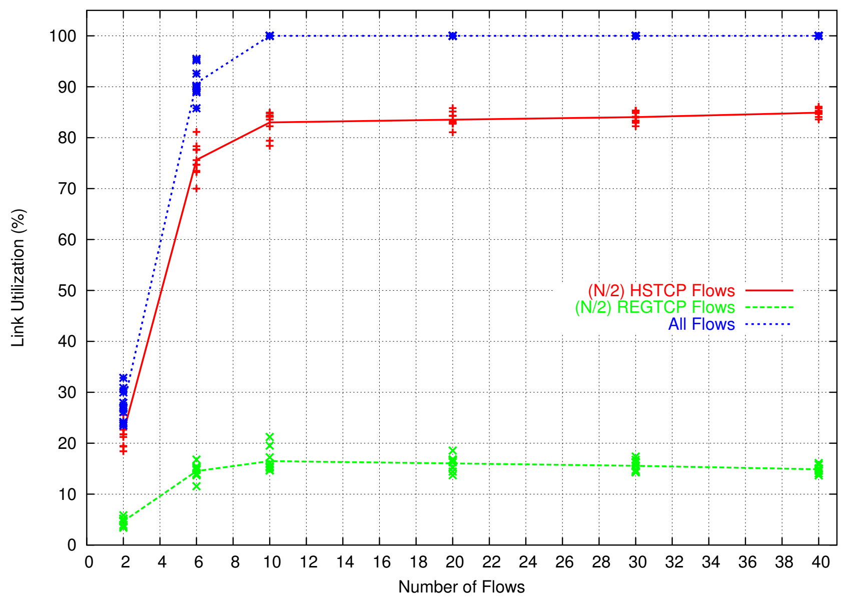
(b) DT

Figure 6.22: Congestion Event Rate - Constant Link Loss Rate of 10^{-5} - Homogeneous Flows

These graphics show the influence router queue management has on the behavior of link utilization for each type of flow. While for RED, the link utilization for the HSTCP



(a) RED



(b) DT

Figure 6.23: Aggregated Link Utilization - Constant Link Loss Rate of 10^{-5} - Heterogeneous Flows

flows decreases as the total number of flows increases, for DT, the same link utilization stays constant or even slightly increases, as the number of flows increase.

The aggregated relative fairness for the third flow set is depicted in Figure 6.24. It shows the ratio between the amount of bandwidth used by all HSTCP flows and the amount of bandwidth used by all the remaining REGTCP flows.

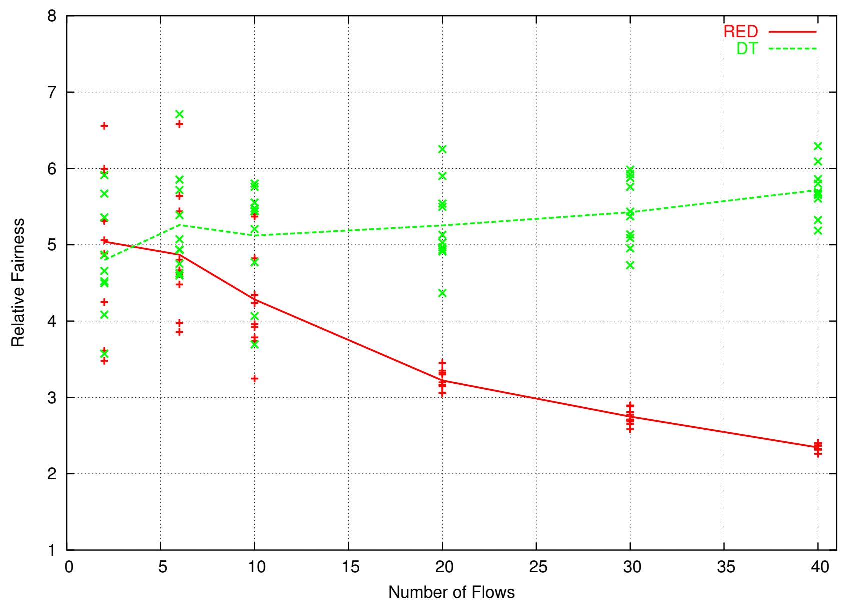


Figure 6.24: Aggregated Relative Fairness - Constant Link Loss Rate of 10^{-5} - Heterogeneous Flows

The last result presented in this experiment is the amount of aggregated bandwidth stolen from all the REGTCP flows when they are deployed together with the HSTCP flows, Figure 6.25.

From Figure 6.24 and 6.25 we see that the HSTCP flows are getting more bandwidth share as the number of flows increase, when DT is used. RED router queue management presents the opposite result.

6.7 Long Term Simulation

This experiment illustrates the interaction of 1 HSTCP flow and 1 REGTCP flow over a long period of time. We ran the experiment for a period of 3600 seconds or 1 hour, using RED router queuing policy. The Figure 6.26 shows the behavior of the congestion window of both flows during this period of time.

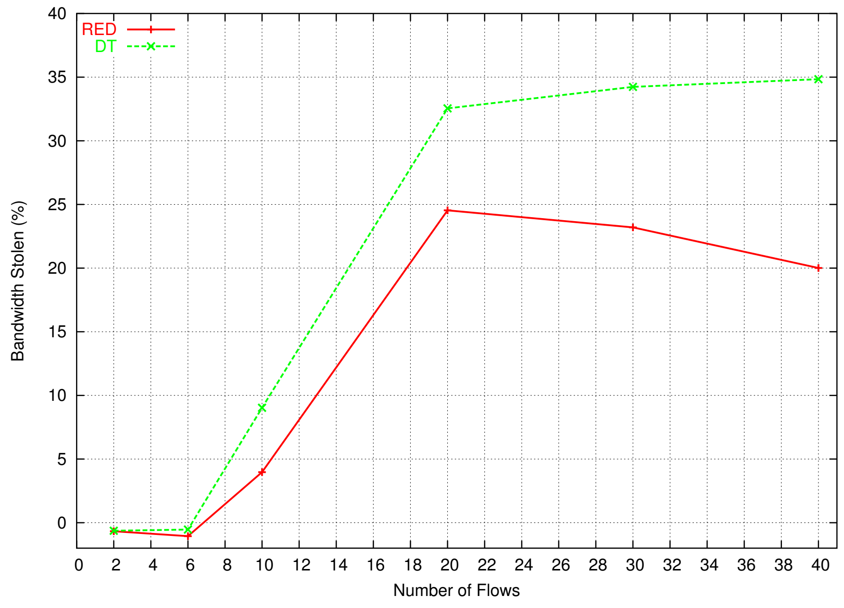


Figure 6.25: Aggregated Bandwidth Stolen - Constant Link Loss Rate of 10^{-5}

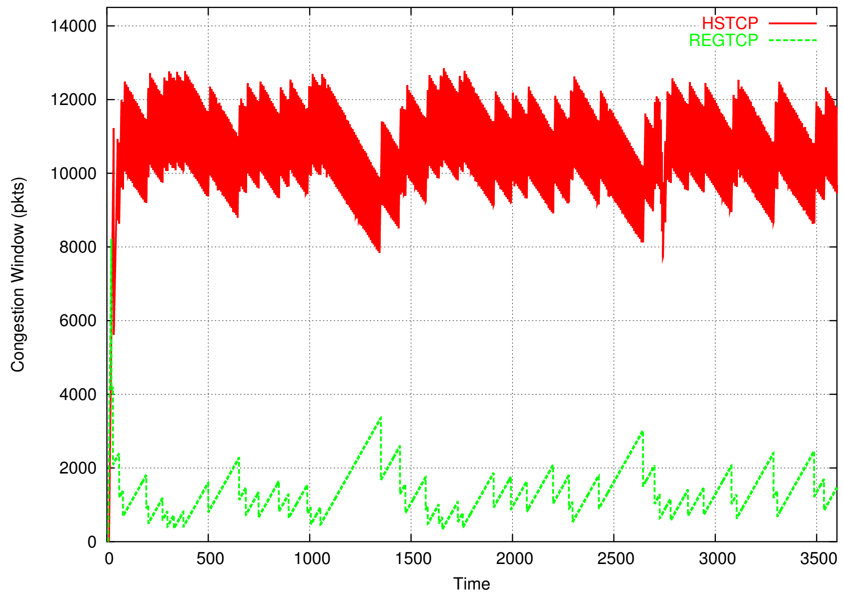


Figure 6.26: Long Term Simulation - 1 Hour - RED

6.8 Parallel Streams Transfer

The focus of this set of experiments was to verify the theoretical performance of parallel TCP streams when they operate in several levels of packet loss. We used the simulator error model to simulate losses in the bottleneck link. This loss model was set to drop a

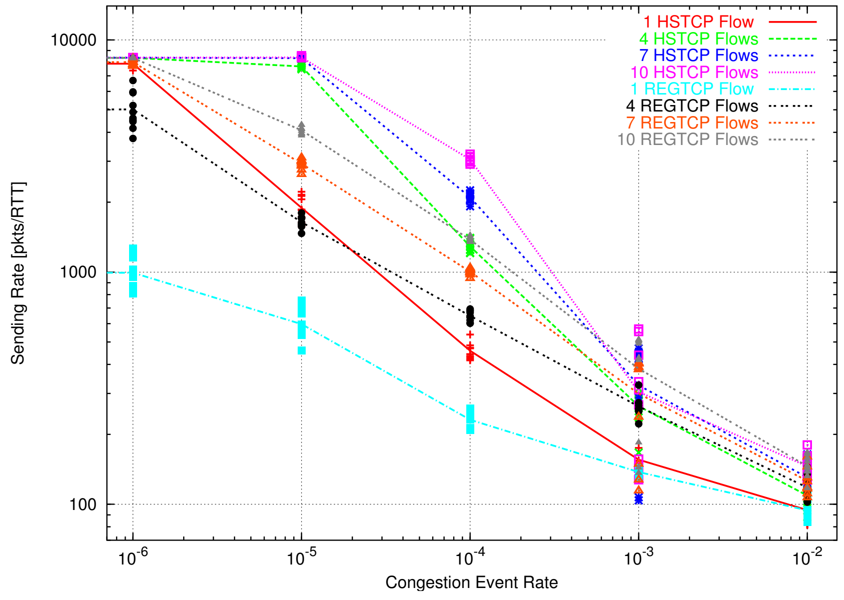


Figure 6.27: Response Function - Parallel Streams Transfer - RED

packet with a defined average drop rate. The loss rates used were 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, and 10^{-2}. We used parallel TCP streams containing 1, 4, 7 and 10 flows. The first set of flows have only REGTCP flows and the second set has only HSTCP flows. The results are presented separately for each number of flows. These sets of flows were run with RED and DT queuing policy in the routers. Each simulation ran for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another just by the random number generated to begin the simulation each time.

Figure 6.27 presents the performance of each transfer in terms of its sending rate. The unit used here was packets/RTT and RED router queue management was used. The use of DT presents similar results.

Figure 6.28 and 6.29 were added here to present the theoretical performance expected from the use of parallel TCP stream, and their comparison with the theoretical performance of 1 HSTCP flow.

The theoretical performance of parallel TCP streams over a range of packet loss rates follows the equation presented in [25], for the condition of this experiment (MSS = 1500 bytes, RTT= 100 ms, C = 1, and packet losses impacts parallel streams to the same extent). For the response function of HSTCP was used the equation defined in [18]. Figure 6.29 presents the expected fairness for parallel TCP streams and 1 HSTCP flow relative to 1 standard TCP stream.

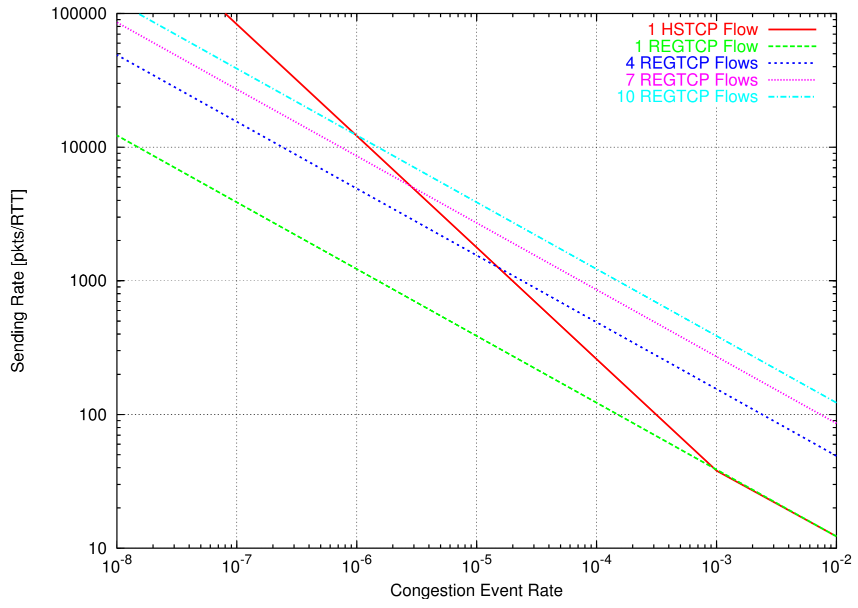


Figure 6.28: Theoretical Response Function - Parallel Streams Transfer

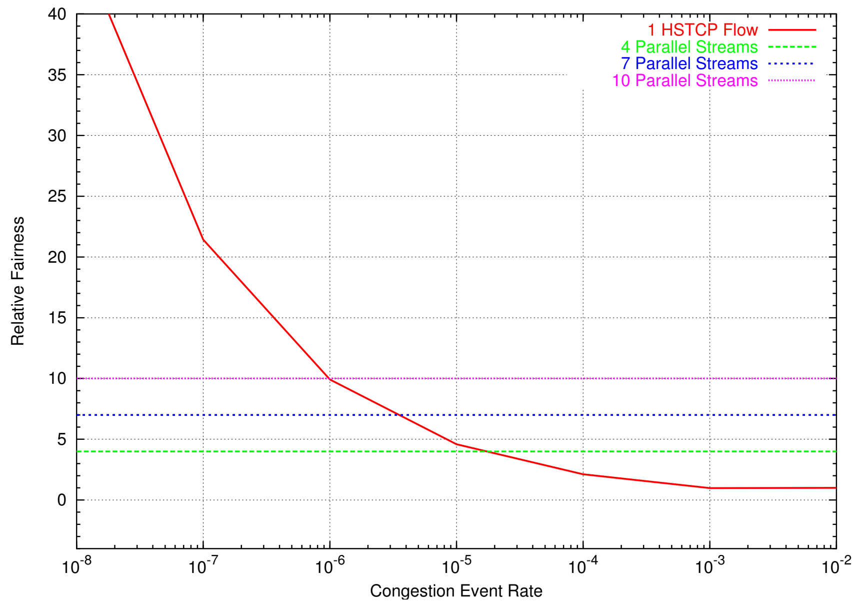


Figure 6.29: Theoretical Per Flow Relative Fairness - Parallel Streams Transfer

6.9 Parallel Streams on Lossy Link Condition

The focus of this set of experiments was to observe the impact of Parallel Streams over long-lived REGTCP flows and to compare it with the impact of HSTCP over the same long-lived REGTCP flows; when both are submitted to systemic losses (losses not due to congestion). We used the simulator error model to simulate losses in the bottleneck link. This loss model was set to drop a packet with a defined average drop rate. The loss rates used were 10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}, and 10^{-2}. We used two sets of flows to develop this experiment. The first set contains 10 REGTCP flows (representing the long-lived flows) and also 1, 4, 7, 10, 20 or 30 parallel streams. The second one is formed by the same 10 REGTCP flows of the first set and one HSTCP flow. Each set of flows was run with the RED and DT queuing policy in the routers. Each simulation ran for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another just by the random number generated to begin the simulation each time. The results show the performance of the selected metrics in this experiment.

The two graphs in Figure 6.30 present the aggregated link utilization for the 10 long-lived REGTCP flows, when RED and DT are deployed, respectively. There is also present in these graphs the performance of 10 long-lived REGTCP flows when no parallel streams are present.

These graphs show us that the performance of 10 long-lived regular TCP flows is not impacted by the use of parallel streams until the total link bandwidth is reached. After this point, their performance is compromised. The same happens when one HSTCP is used.

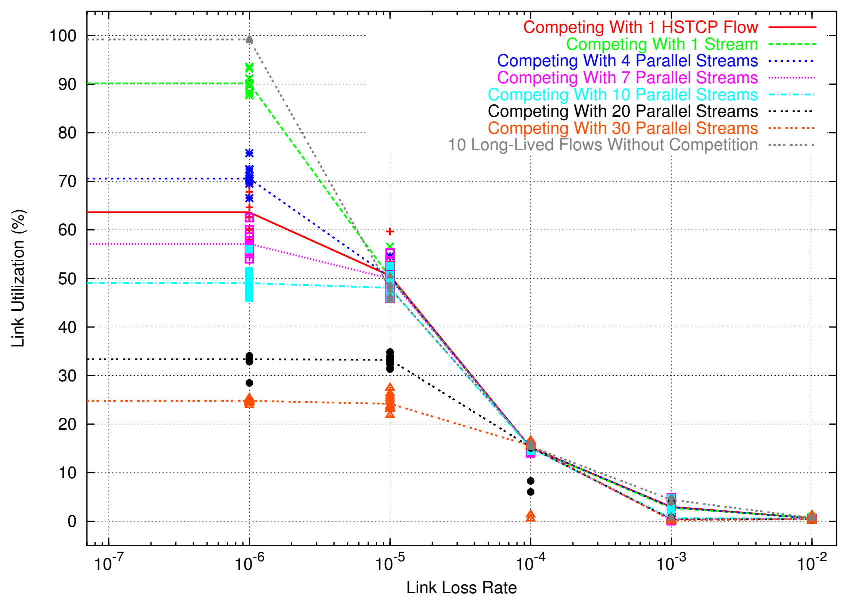
The information describing the impact of the use of parallel streams is complemented by the graphs in Figure 6.31. They show the performance of the parallel streams and the HSTCP flow in this context.

The important information available is that, after the bandwidth limit has been reached, the link utilization stays constant, according to the number of parallel streams. The same happens to the HSTCP flow. Another important aspect presented here is that, for link loss rates greater than 10^{-3}, the performance are really bad for all of the schemes.

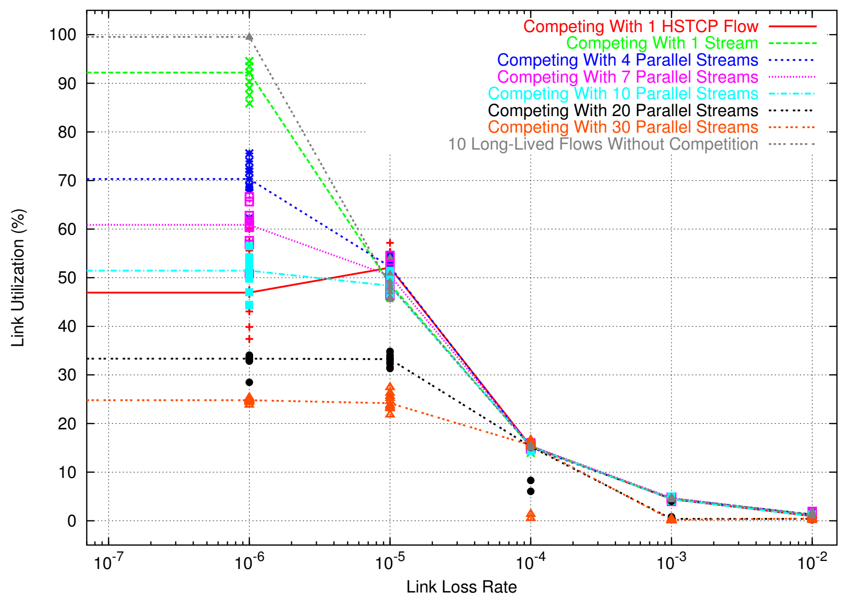
The next two graphics in Figure 6.32 present the aggregated congestion event rate for the first and second set of flows, when RED and DT are deployed.

Finally we present the per flow relative fairness. The intention here is to show the competition that a parallel stream transmission represents for a single long-lived regular TCP flow. The amount of link bandwidth used for the parallel stream transmission is divided by the amount of link bandwidth used by one of the 10 long-lived streams. The same procedure is used for the case of the transmission using one HSTCP flow. The results are presented in Figure 6.33.

It is clear that, when parallel streams are deployed, the relative fairness is almost constant over a wide range of link loss rates. This behavior only changes when there is a heavy packet loss rate. In contrast, the relative fairness when HSTCP is used is not constant and has a wide range of values.

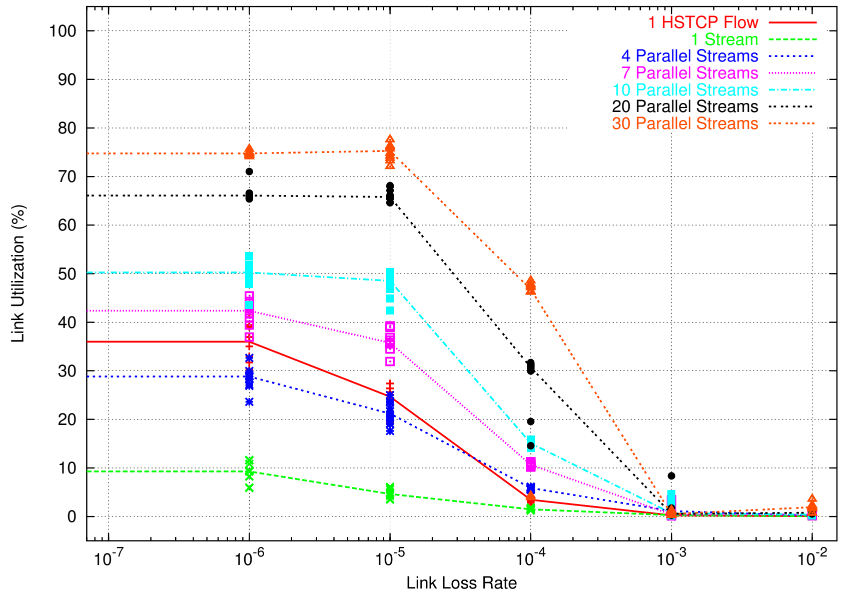


(a) RED

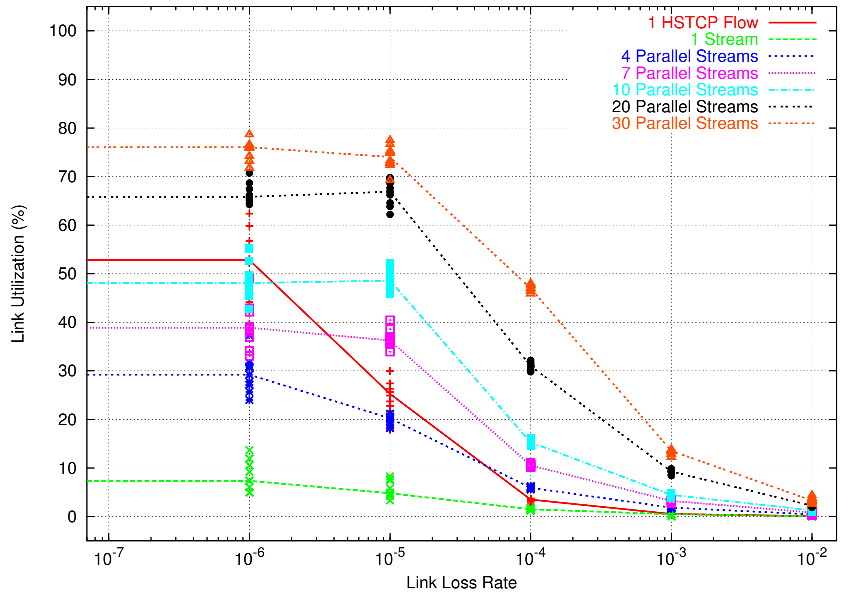


(b) DT

Figure 6.30: Aggregated Link Utilization of 10 Long-Lived Regular TCP Flows - Parallel Streams on Lossy Link Condition



(a) RED

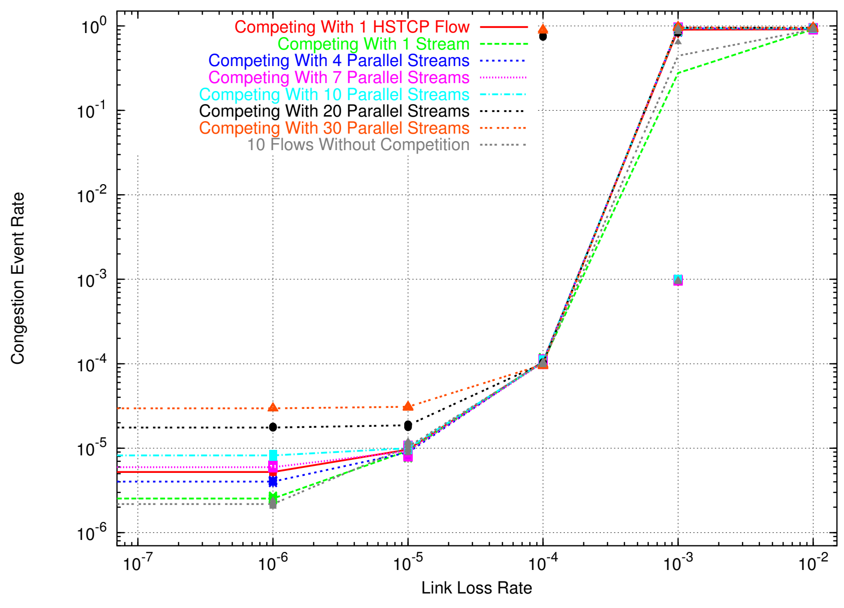


(b) DT

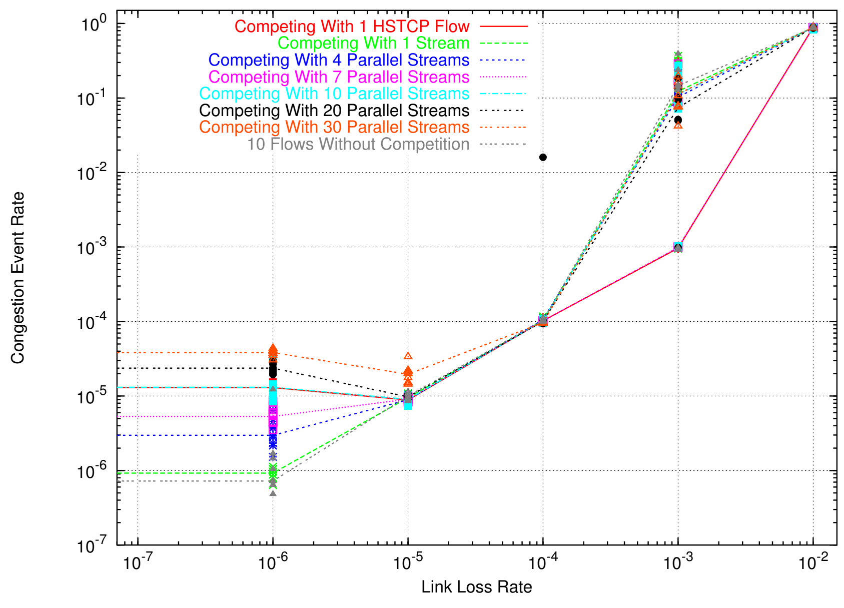
Figure 6.31: Aggregated Link Utilization of Competing Parallel Streams - Parallel Streams on Lossy Link Condition

6.10 Parallel Streams on Bursty Traffic Condition

The aim of this set of experiments is to observe the impact of Parallel Streams on long-lived REGTCP flows and to compare it with the impact of HSTCP over the same



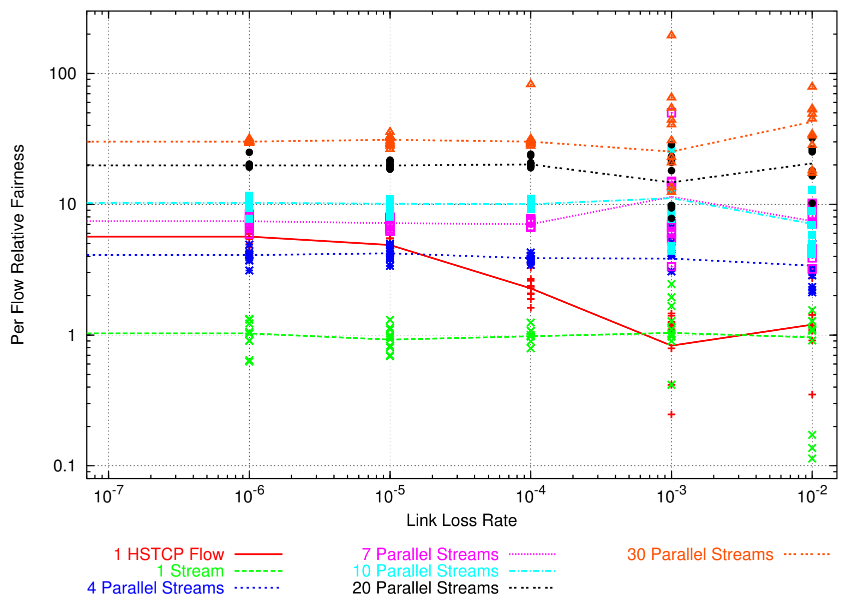
(a) RED



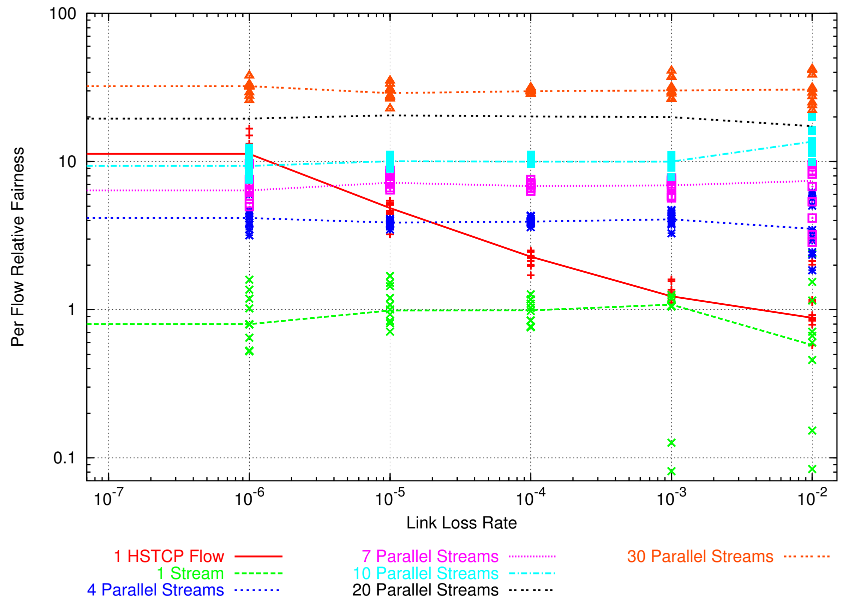
(b) DT

Figure 6.32: Congestion Event Rate - Parallel Streams on Lossy Link Condition

long-lived REGTCP flows; when they are submitted to bursty traffic. This bursty traffic is composed of short-lived standard TCP flows running for a few seconds, in such a way that



(a) RED



(b) DT

Figure 6.33: Per Flow Relative Fairness - Parallel Streams on Lossy Link Condition

they only run during their Slow-Start phase. As the Slow-Start phase has an exponential growth, the bursty flows have a considerable impact in the long-lived flows. We used 0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 60 and 70 bursty flows, randomly distributed during all the simulation time.

We used two sets of flows to develop this experiment. The first set contains 10 REGTCP flows (representing the long-lived flows) and also 1, 4, 7, 10 or 20 parallel streams. The second one is formed by the same 10 REGTCP flows of the first set and one HSTCP flow. Each set of flows was run with RED and DT queuing policy in the routers. Each simulation ran for 300 seconds, and each one was repeated 10 times. The line crossing the points represents the median of these 10 simulations. Each repetition differs from another just by the random number generated to begin the simulation each time. The results show the performance of the selected metrics in this experiment.

The first two graphics in Figure 6.34 present the aggregated link utilization for the 10 long-lived REGTCP flows, when RED and DT are deployed, respectively. Also presented in these graphs is the performance of 10 long-lived REGTCP flows when no parallel streams are present.

Information regarding the impact of the use of parallel streams is completed by graphics in Figure 6.35. They show the performance of the parallel streams and the HSTCP flow in this context.

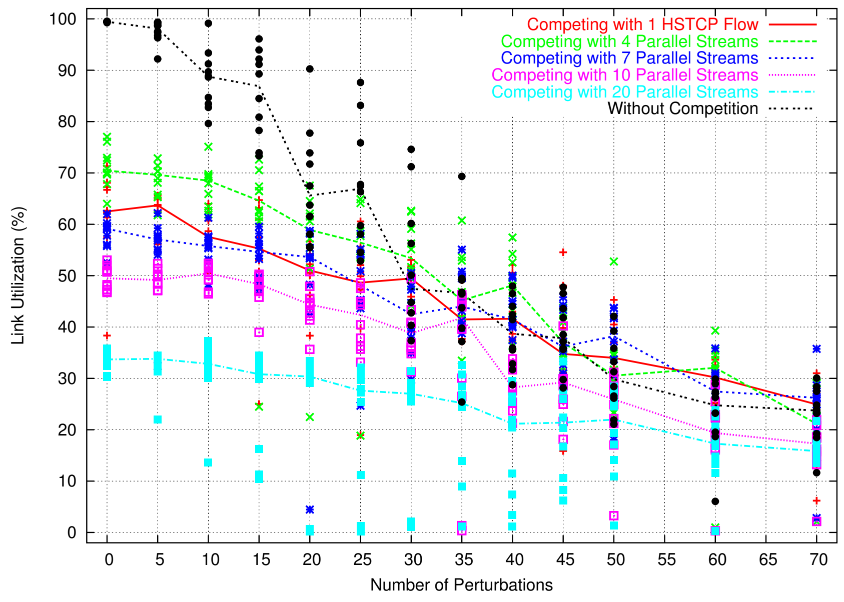
Two facts are revealed through these graphs. The first one is that the performance of parallel TCP streams tends to decrease as the number of perturbations increase. This is more evident when RED is deployed than when DT is used. The second fact is that the performance of HSTCP is much less sensitive to this environment, and even improves as the number of perturbations increases.

In the following we present the results for per flow relative fairness. The intention is to how what represents a parallel stream transmission in terms of competition, for a single long-lived regular TCP flow, when both are submitted to bursty traffic. The amount of link bandwidth used for the parallel stream transmission is divided by the amount of link bandwidth used by one of the 10 long-lived streams. The same procedure is used for the case of the transmission using one HSTCP flow. The results are presented in Figure 6.36.

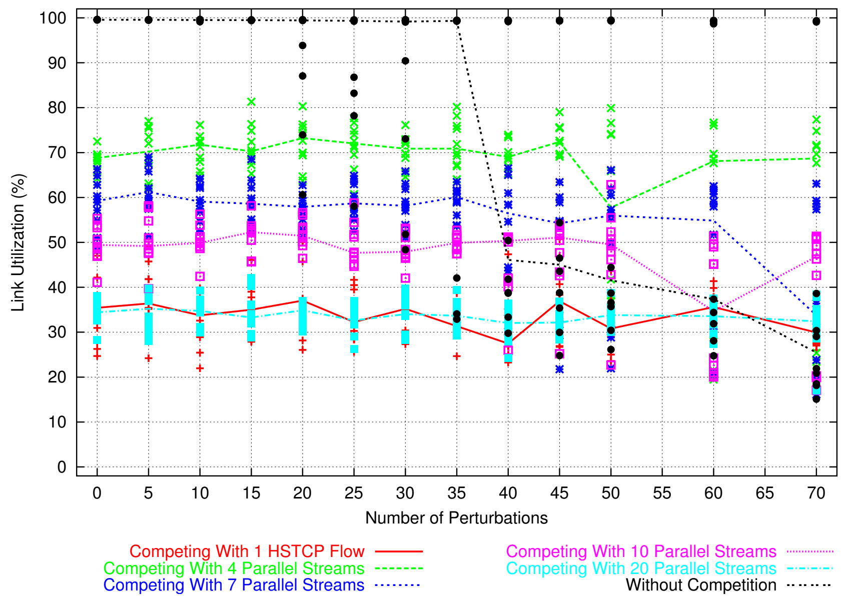
We observe in these graphics that when RED is used the relative fairness increases as the number of perturbations increase, but this behavior is not clear when DT is deployed. In both cases the ratio between the bandwidth used by HSTCP and the bandwidth used by one of the 10 long-lived flows could spread over a wide range of values.

The aggregated bandwidth stolen from the long-lived TCP flows when they are deployed together with HSTCP flows and parallel streams is presented in Figure 6.37.

The important message in this graphic is that the amount of bandwidth stolen from the 10 long-lived TCP flows decreases as the number of perturbations increases, independent of the type of router queue management, and independent of the scheme of bulk data transfer used.

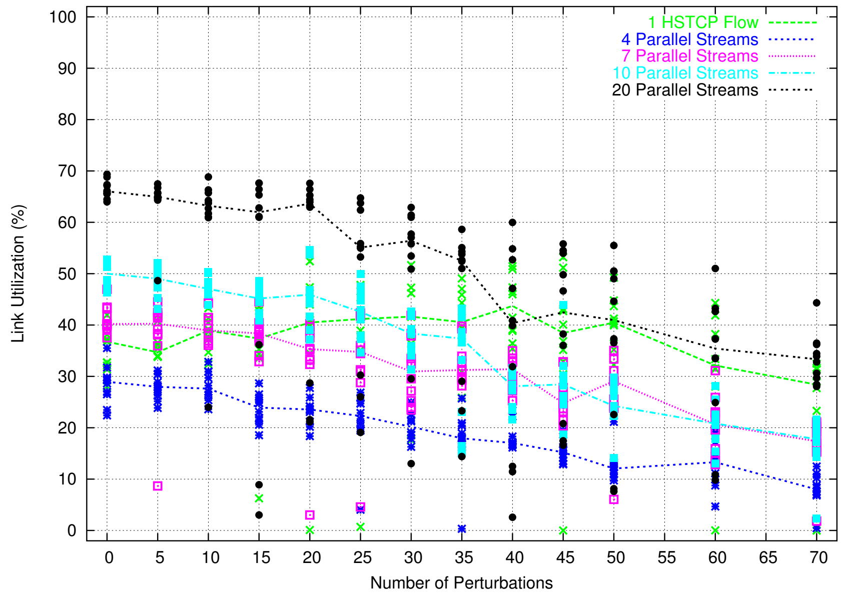


(a) RED

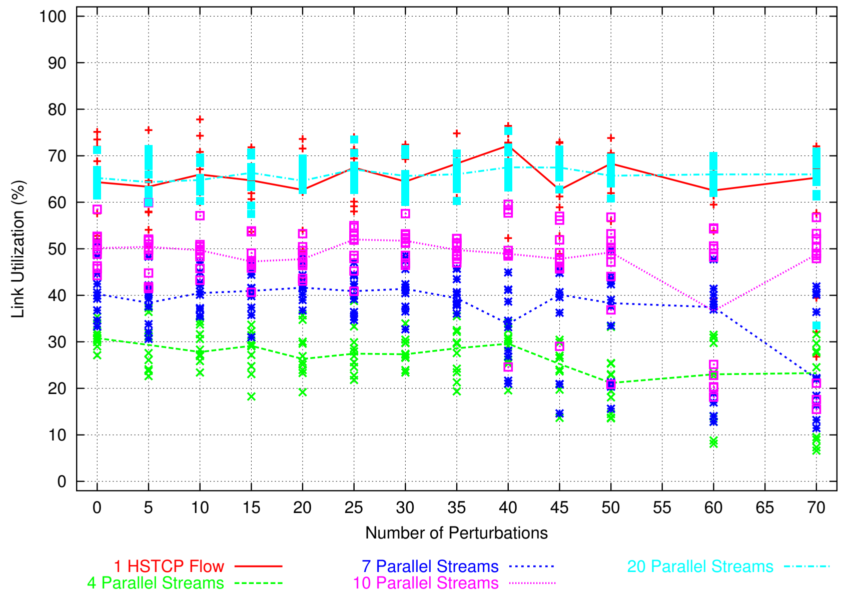


(b) DT

Figure 6.34: Aggregated Link Utilization of 10 Long-Lived Regular TCP Flows - Parallel Streams on Bursty Traffic Condition



(a) RED

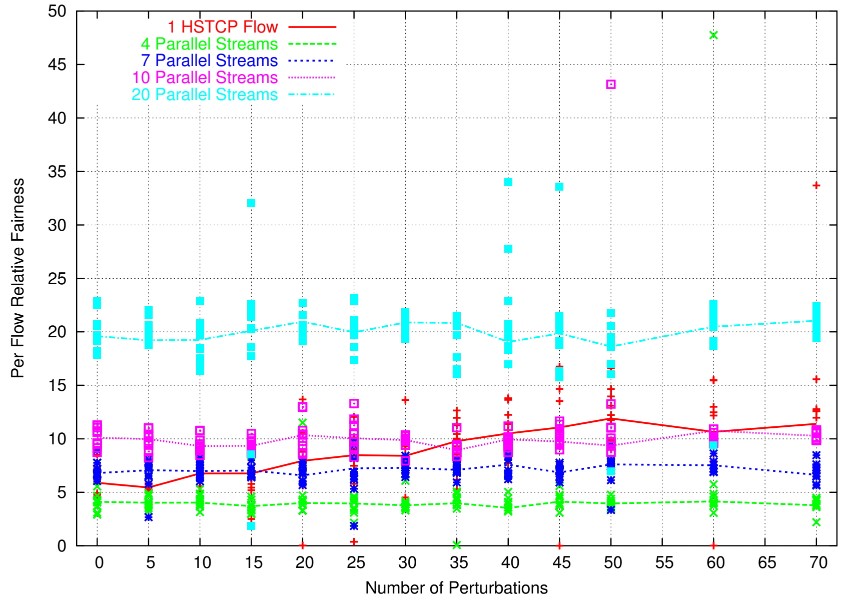


(b) DT

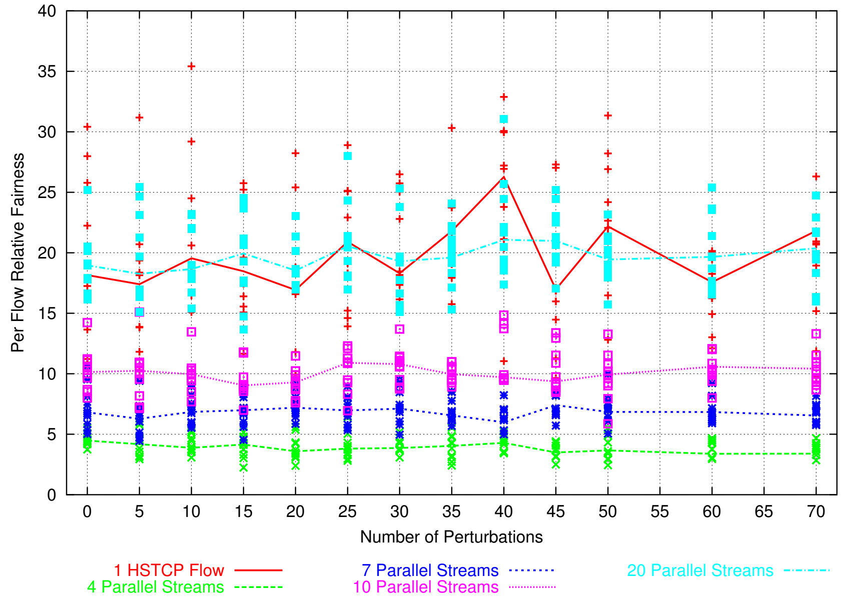
Figure 6.35: Aggregated Link Utilization of Competing Parallel Streams - Parallel Streams on Bursty Traffic Condition

6.11 Slow-Start

Even though this is not a central part of this work, we developed an experiment with a focus on the slow-start phase, using 1 HSTCP flow. The reason for this was to try



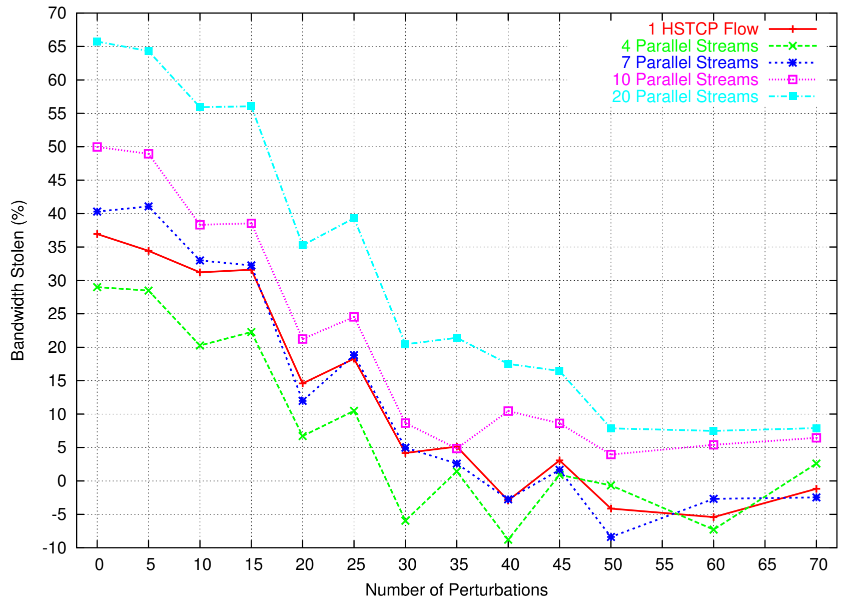
(a) RED



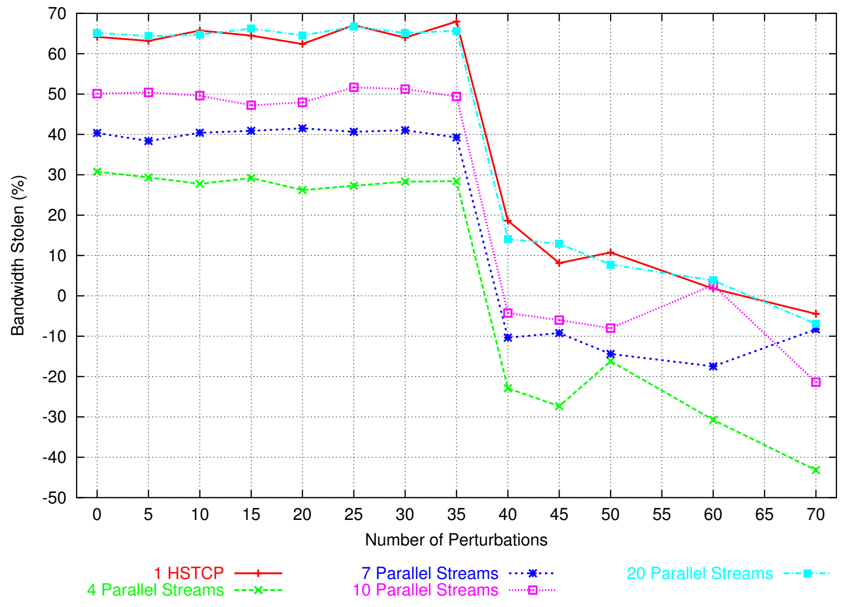
(b) DT

Figure 6.36: Per Flow Relative Fairness - Parallel Streams on Bursty Traffic Condition

to understand the effect that some parameters in the Slow-Start have over the steady-state phase of a transmission. We used two algorithms to accomplish this objective. The first one



(a) RED



(b) DT

Figure 6.37: Aggregated Bandwidth Stolen - Parallel Streams on Bursty Traffic Condition

was the standard slow-start algorithm and the second algorithm was the modified slow-start algorithm, proposed in [17] for large congestion windows. The parameter MAX_SSTHRESH for the modified slow-start was set to 10, 100 and 1000. Each algorithm was simulated one time for 100 simulation seconds, using the default parameter for bandwidth and link delay.

Figure 6.38 presents the evolution of the congestion windows using the standard and the modified Slow-Start algorithms. Figure 6.39 presents the evolution of sequence number for the standard slow-start, when used by 1 HSTCP with different Slow-Start, as used on the graph before. Finally, the table Table 6.1 present the amount of packets dropped for the simulation in Figure 6.38, in the first half of the simulation.

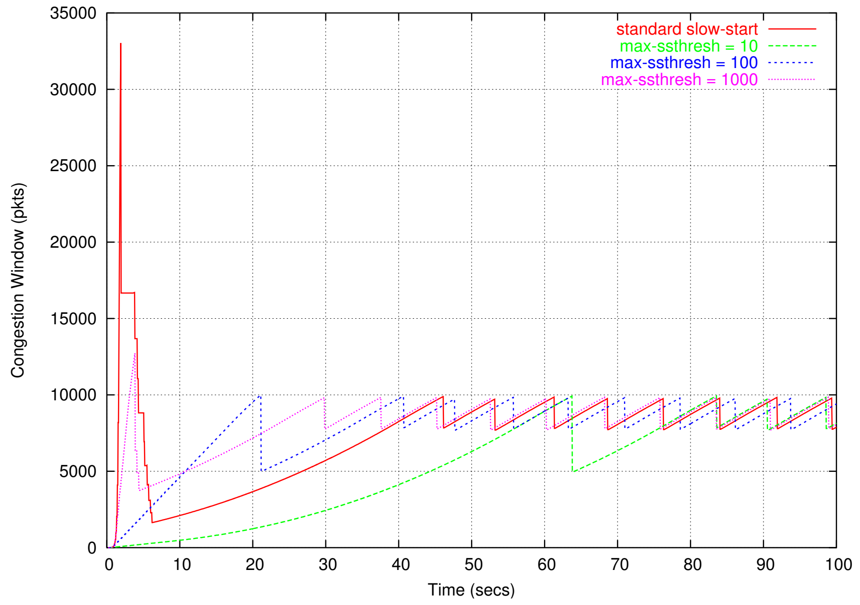


Figure 6.38: Slow-Start Variation with MAX-SSTHRESH

MAX-SSTHRESH	PKTS LOST/MARKED	TIME OF CHANGE	CWND IN CHANGE
0	27274	1.30	33032.00
10	1	63.75	9919.98
100	1	20.75	9979.66
1000	58	3.55	12726.20

Table 6.1: Slow-Start Comparison

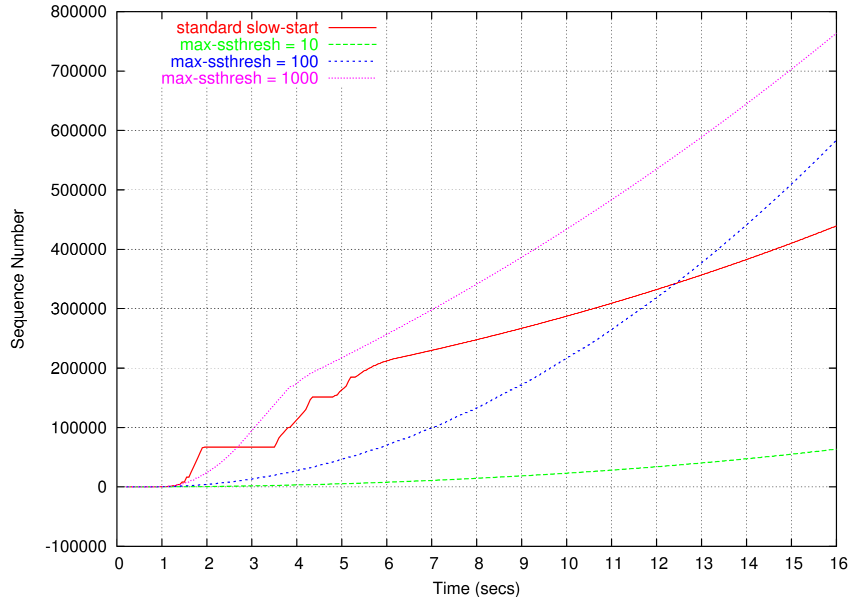


Figure 6.39: Effect of Packet Loss in Slow-Start

Chapter 7

Discussion of the Results

This section contains a discussion concerning the results presented in the previous chapter. The main reasons behind the results are analyzed and questions proposed in this work are discussed here.

7.1 Organization of Results

The results are presented as follows. First we present a comparison of the performance of HSTCP and REGTCP for the network conditions defined in these experiments. Of special interest is the performance of HSTCP in situations where REGTCP shows poor performance.

The second part is dedicated to understanding the interactions among flows of both HSTCP and REGTCP. Fairness issues are presented and discussed in this part, as well as, an analysis of the possibility of deploying HighSpeed TCP together with the Standard TCP protocol.

The third part provides a discussion of the effect of router queue management. Buffer management has an important influence on the performance metrics of both protocols and also on their interactions.

The fourth part presents an examination of HighSpeed TCP as a candidate method for bulk data transfer. We explore how many REGTCP flows a single HSTCP flow can substitute for, and compare HSTCP performance with other method of bulk data transfer in different network conditions.

The final part addresses some other issues involving Slow-Start and TCP implementation that arose during the development of this study, but are not directly related to the subject of this study.

7.2 Questions for the Deployment of HighSpeed TCP

7.2.1 What is the behavior of HighSpeed TCP in situations where Regular TCP underperforms?

Ideal Condition

The environment used for the tests in the first network condition places no restrictions to keep the set of REGTCP flows from reaching the maximum link utilization, except the time available to run. The time for the experiment run is long enough to expect good performance. Even though this is true, a small number of REGTCP flows are unable to completely use the available bandwidth. With less than 10 flows, the set of REGTCP flows does not use the full bandwidth, in this network condition. Only with a higher number of REGTCP flows is the link capacity used. Figure 6.2 presents this situation. If there is systemic loss in the path, the situation becomes worse, as seen in Figure 6.21.

The reason for this poor performance is related to the bandwidth delay product of the bottleneck link, and the conservative increase of CWND in the REGTCP Congestion Avoidance phase. Given that, for each segment ACKed during the Congestion Avoidance phase, the CWND is increased by 1/CWND, and the interval between each increase is one RTT, the evolution of the congestion window is slow. This slow growth leaves the link with a low level of utilization during a significant period of time. This situation is presented on Figure 6.1(a). Even though regular TCP will eventually reach the bandwidth limit, it is clear that during a large amount of time, the link will be under-utilized.

It is important to point out that other restrictions to TCP reaching a high throughput, such as limited TCP buffers, reduced network buffer capacity and large packet loss in Slow-Start, are not present in this case.

At the gigabit speeds, used in this network scenario, the performance of REGTCP is latency limited, rather than bandwidth limited. The latency is caused by the speed of light and cannot be decreased. The other possible limit is determined by the TCP window size [56]. As shown here, the standard TCP protocol limits the performance by its slow increase rate of the congestion window. This slow dynamic response incurs other performance problems that will be explained later.

Unlike REGTCP, a few HSTCP flows are enough to fill the pipe, as can be seen in Figure 6.2. This is a direct consequence of HSTCP having a faster increase rate of CWND than the increase rate of REGTCP during the Congestion Avoidance phase. As previously presented in the chapter about HighSpeed TCP Fundamentals, the parameter used to increase the congestion window is a function of the present congestion window. The larger the congestion window, the greater the factor.

As seen in the results, HSTCP makes better use of a high bandwidth link with high delay. It avoids a long period of exploratory growing in the Congestion Avoidance phase and reaches a high-level steady-state in a short period of time.

The difference in behavior of both protocols is also visible in Figure 6.3(a). There is a clear difference between the congestion event rate produced by each protocol in the same situation. This difference can be understood as the difference in the aggressiveness. As the HSTCP ramps up faster than REGTCP, the probability that one of HSTCP flows reaches the bandwidth limit, and consequently produces a congestion event, is higher than for REGTCP. Even with a large number of flows (30 or 40), this characteristic is present. This fact also suggests that, as REGTCP is slow to adjust its congestion window in a high delay environment, when a congestion event happens, the probability that some of its flows reach the limit is lower than for HSTCP protocol.

It is important to observe that the congestion event rate for one HSTCP flow cannot be lower than 10^{-6}, as can be seen in Figure 6.3(a). This is due to the fact that this is the level at which it reaches the maximum transmission rate, for the conditions in this experiment. It is interesting to note that this is the crosspoint of the HSTCP theoretical response function line, and the line of link bandwidth. This is seen in Figure 3.1.

Lossy Link Condition

In the second experiment, we studied the impact of packet loss not caused by a router buffer overflow but by a faulty transmission, also called systemic loss. The dominant understanding is that packet loss is caused exclusively by routers dropping packets when queues overflow, and it is interpreted by TCP as an indication of network congestion between a sender and a receiver. But packet losses may have other sources. In [57], the authors show that between 1 packet in 1100 and 1 packet in 32000 fails the TCP checksum and the packet is dropped, even when the data-link CRC checksum passes.

Packet loss may be due to random factors other than network congestion, such as intermittent hardware faults [2]. An example source of packet loss not due to congestion is described in [10], in which a large amount of packet loss in a cable modem was caused by a hardware bug. The authors in [46] found that ATM cell drops due to hardware problems limited TCP performance over OC-12 links. Losses not related to buffer overflow also happen in satellite connections or other forms of wireless communication.

Given this evidence, the belief that packet losses are caused only by queue overflows cannot be supported. There are many pieces of network equipment and carriers involved in a wide-area network data transmission, a bug or error in any of these pieces of equipment may corrupt or drop a packet.

With such a potentially high systemic losses, this experiment studied the effect of several packet loss rate levels on the two types of TCP.

The resulting link utilization of the set containing only REGTCP flows shows an impressive performance loss when the link loss rate increases. This prevents REGTCP from making reasonable use of the bandwidth available, as depicted in Figure 6.8. When the packet loss rate is greater than 10^{-6}, REGTCP is unable to use the link fully. This situation happens for HSTCP only after 10^{-5}. The performance of the flow set using HSTCP is a bit better in the range from 10^{-6} to 10^{-3}. The link loss rate of 10^{-3} is the point set for the Low_Window in the HSTCP parameters (a congestion window of 31 packets represents a packet loss rate of approximately 0.001). Beyond this loss rate, the performance of HSTCP flows is equivalent to REGTCP flows, by design.

It is interesting to notice that the congestion event rate for HSTCP flows cannot be lower than 10^{-5}, as can be seen in Figure 6.9(a). Around this value, there is a "knee" in link loss rate. The reason for this knee is that the influence from the link losses becomes lower than the influence of the real congestion induced by the limit on the available bandwidth. In the case where RED router queue management is used, below the link loss rate of 10^{-5}, the number of congestion events produced by ECN-marked packets is greater than the congestion events produced by packet losses.

Figure 6.8 also shows that a link loss rate between 10^{-5} and 10^{-4} prevents REGTCP from making a reasonable use of the link bandwidth available (less than 50% in this case). In this range, the HSTCP flows are able to use almost the double of bandwidth that REGTCP flows use, with the parameters used in HSTCP flows configuration.

Another way to understand the performance of HSTCP in relation to REGTCP is to observe their link utilization when a fixed link loss rate is set. Figure 6.21 shows that HSTCP protocol needs only 6 flows to achieve full link utilization, while REGTCP protocol only reaches this performance with 20 or more flows. This particular situation is closer to the reality, since it is hard to find a path without some systematic loss.

The same knee presented in Figure 6.9(a) is also present in Figure 6.22(a). The set of REGTCP flows has its knee in the congestion event rate only after 30 flows, far beyond the 2 flows for the set of HSTCP flows.

Bursty Traffic Condition

Network traffic can present burstiness, because of its inherent self-similar nature [40]. This condition is of particular importance in our analysis.

In a typical network, TCP optimizes its send rate during Slow-Start by releasing increasingly large bursts (or windows) of packets, one burst per round-trip time, to the receiver until it reaches its maximum window size. At this point it has reached the full capacity of the network. In a network with a high bandwidth delay product, however, TCP's maximum window size may be larger than the queue capacity of some of the network's intermediate routers. Larger windows overload such router queues, and the routers begin to drop packets.

This TCP behavior has several effects on the network elements as well as on the other traffic present in the same link:

- Increased queuing delay: The bursty nature of traffic leads to buffering of packets at the intermediate routers. However, buffering delays could also depend on the congestion level, queuing and scheduling policies. Larger queues at the routers may introduce additional delays in the TCP flows and increase their RTT.
- Jitter (variable delay): In some networks, delay variation mainly occurs due to queuing from bursty traffic.
- Make other traffic flows also bursty: The burstiness of one TCP flow can cause the other TCP flows to become bursty when they share a common queue at an intermediate router. The effect can be catastrophic if these bursts get synchronized. According to global synchronization effects described in [63], packet losses due to buffer overflow can be synchronized causing all the TCP flows to back off simultaneously and under utilize the bandwidth.
- Low throughput levels: The burstiness of TCP results in packet drops arising from queue overflows or an increase in RTT due to queuing delays. This leads to low throughput levels for TCP flows. This along with other side effects, like synchronized window evolution, could waste bandwidth and also produce bursty packet losses.
- Unfair sharing: The bursty nature of TCP may result in unfair competition traffic between streams caused by the queuing bottleneck.

The effects of having bursty TCP traffic competing for the link were significant for all sets of flows.

The set of REGTCP flows presented a continued decrease in the link utilization as the number of bursty perturbations increased, this can be seen in Figure 6.13(a). The explanation is that the congestion window of the REGTCP flows is cut in half each time a congestion event happens, it does not recover the previous throughput because the increment in the congestion window is only 1/CWND each RTT. When there is a high link bandwidth, this leads to a low link utilization.

This poor link utilization is not because the perturbations are using a large share of the link bandwidth. Rather, it is due to the bursty nature of these perturbations. It is possible to see that the perturbation flows use less than 10% of the link capacity, when the link utilization for the set of REGTCP flows dropped around 70%.

The number of congestion events for the set of REGTCP flows is low even though the number of bursty perturbations is high. This happens because, as the REGTCP flows decrease their throughput, the probability that some REGTCP flow has a packet in the queue is also low, and the majority of packets lost will belong to the perturbing flows.

The set of HSTCP flows decreases its link utilization also, but smoothly and slowly as the number of perturbations increases, as can be seen in Figure 6.13(a). It is clear that the HSTCP flows lose less bandwidth than the set of REGTCP flows does, so it is more resilient to this kind of traffic and recovers faster from this interference.

The amount of bandwidth utilized by a perturbation when competing against the HSTCP flow set is slightly inferior to the bandwidth used when competing against a REGTCP flow set, as seen in Figure 6.13(a). This suggests two things. First, the amount of bandwidth lost by the REGTCP flows is again not due to the perturbation flows using the bandwidth. Second, the perturbation flows have to compete hard with the HSTCP flows to use the bandwidth available, or the HSTCP flows resist them better.

7.2.2 Is it possible to use HighSpeed TCP together with Regular TCP and maintain an acceptable fairness?

The HSTCP flows presented better performance than the REGTCP flows for high bandwidth, long delay links, as seen in the previous question. It is clear that HSTCP is more aggressive than REGTCP, which contributes to this performance. On this topic, an interesting question arises: how much can the HSTCP flows hurt the performance of the REGTCP flows when they are deployed together?

Fairness is a key point in the acceptance of a protocol or solution in best effort network. This aspect has raised concerns in the network community in the past [15], and makes it difficult for deployment and co-existence with other protocols [25]. If an implementation of a congestion control protocol is much more aggressive in its use of bandwidth than other implementations, it could induce other new protocols to be more aggressive as well.

A flow is *TCP-compatible* if it is responsive to congestion notification, and in the steady-state uses no more bandwidth than a conformant TCP running under comparable conditions of drop rate, RTT and MTU. As mentioned before, to a certain degree, HSTCP is not "TCP-compatible", but its degree of compatibility changes according to the drop rate perceived by a HSTCP flow.

We studied here the steady-state fairness, which is important to bulk data transfer. Other situations, where fairness is also important, such as in Slow-Start and in some transient conditions, are not discussed here.

Ideal Condition

If there is no external interference, the bandwidth share used by HSTCP flows is higher than that used by REGTCP flows, when both types of flows compete for the same link. The difference in the link utilization can be seen clearly in Figure 6.4(a). The link presents a high level of utilization when both types of flows are competing together. It is noticeable that the amount of the link used by the HSTCP flows decreases as the total number of flows increases. The opposite happens with the REGTCP flows. The reason for this behavior is found in the fact that the HSTCP CWND update is tied to the level of congestion event rate perceived by the flows. The higher number of flows competing for the link bandwidth, the more congestion events happen and the less aggressive the HSTCP flows are. With the decrease in the HSTCP aggressiveness, the REGTCP flows have more opportunity to use the bandwidth available.

The same behavior is observed in the aggregated relative fairness, as seen in Figure 6.6. The disproportion between the link utilizations decreases as the number of flows increases. A broad range of fairness values happen when there is a small number of flows competing for the link. The reason for this could be associated with an early congestion event that reduces significantly the congestion window size of the REGTCP flow. As there are only a few REGTCP flows, this contributes to the overall lower link utilization. The time when a congestion event happens will define the REGTCP flow link utilization, and consequently the relative fairness. The difference can be large, but not enough to prevent REGTCP using part of the link bandwidth.

Figure 6.5 presents the evolution of the congestion event rate for this situation, and shows the increase in the congestion event rate as the number of flows increases.

The percentage of the total bandwidth capacity that the HSTCP flows take from the REGTCP, when they are competing, is expressed in Figure 6.7. Again, the effect of the variation in the congestion event rate is visible in these results.

The results presented before highlight two distinct characteristics of the HSTCP protocol. It is more aggressive at using the bandwidth available, but it decreases its aggressiveness as the congestion event rate increases. This adaptability is very interesting in the context of high speed links. It avoids having a link become idle due to the slow dynamic of standard TCP, and yet does not prevent more standard TCP streams from obtaining a reasonable share from the link. This adaptability is expressed in Figure 6.19(a). In this experiment, it is possible to see how the bandwidth is shared when there is only 1 HSTCP flow, and the number of REGTCP flows competing for the link increases. The HSTCP flow backs off gracefully as the number of REGTCP flows increases. The total link utilization is kept near to 100%, which means that the bandwidth resource is fully utilized.

In the same graph, the intersection point of the lines of link utilization of 1 HSTCP flow and the line of link utilization of the REGTCP flows represents the moment where 1

HSTCP flow is equivalent to (N - 1) REGTCP flows, or they utilize the same bandwidth. For the network conditions of this experiment N = 7, which means that 1 HSTCP flow is equivalent to 6 REGTCP flows.

So far, we have presented the interaction between HSTCP flows and REGTCP flows for a period of time of 300 seconds. It is also important to see their interactions over a longer period of time. Figure 6.26 presents the evolution of the congestion window of 1 HSTCP and 1 REGTCP competing flows, for a period of one hour. It is clear that there is little difference in the interaction. The congestion windows are kept around the same level during the entire period. The large area occupied by the HSTCP line represents the oscillation in the HSTCP congestion window.

Lossy Link Condition

In the lossy link condition, the difference in the link utilization from the HSTCP flows and REGTCP flows decreases with the increase in the number of losses, as expected, see Figure 6.10(a), and the relative fairness decreases with the increase in the link loss rate, as seen in Figure 6.11.

An important question to be considered is how much link bandwidth do the HSTCP flows steal from the REGTCP flows, and where does this happen. The answer is in Figure 6.12. The conclusion is that, with a link loss rate higher than 10^{-4}, the REGTCP flows are limited by systemic loss, instead of having some performance loss due to the use of HSTCP. With link loss rates lower than 10^{-4}, the HSTCP flows begin to steal bandwidth from the REGTCP flows. This change occurs because the link utilization is close to the physical bandwidth limit, as seen in Figure 6.10(a). Beyond this point, the HSTCP flows are directly competing with the REGTCP flows for more bandwidth. This turning point may change according to the link capacity and the number of flows competing for the link. This result is similar to the one found in the literature [25].

Bursty Traffic Condition

As previously mentioned, the bursty traffic hurts the performance of both flow sets. However, the major difference here is the performance of the REGTCP flows, as depicted in Figure 6.16(a). The drop in the link utilization now is lower than when the REGTCP flows are alone suffering the perturbations, as can be seen in Figure 6.13(a). A possible explanation for this behavior is that the REGTCP flows already have a low throughput, because they are competing with the HSTCP flows. The relative fairness for this particular experiment is kept relatively constant, as seen in Figure 6.17.

It is also possible to know how much bandwidth the HSTCP flows steal from the REGTCP flows. The answer is seen in Figure 6.18. When competing against HSTCP with bursty traffic, the performance of the REGTCP flows is relatively constant. This produces the result that when the level of perturbation in the link increases, the difference for the REGTCP flows competing with the other REGTCP flows and competing against the HSTCP flows, become smaller. The conclusion is that the bursty traffic has little influence on the amount of bandwidth that the HSTCP flows steal from the REGTCP flows, and it also has little influence on the fairness.

7.2.3 What is the effect of the router queuing policy (RED or DT) on the performance of HighSpeed TCP and on the fairness between HighSpeed TCP and Regular TCP?

The performance and fairness of HSTCP cannot be completely understood without identifying the influence that the router queue management scheme has over it. In this study, we verify the difference in the use of the two queue management schemes: DT and RED (ECN is active when RED was deployed). Even though there is no intention to perform a deep study of the issues of router queue management, its influence is strong enough to warrant some attention. These queuing policies were chosen because they are widely known, researched and deployed.

Router buffers are an essential element for a packet-switched network. They absorb burst arrivals of packets and reduce potential losses. The larger the buffer, the higher the capacity it has to absorb large bursts. However, it builds up load and increases queuing delays.

DT is the simplest way to perform queue management in the router. It manages the length of the queue using a FIFO (First In First Out) scheme. In this scheme, each new packet that arrives at the queue input port is discarded when the queue buffer space is full. Van Jacobson proposed as early as 1988 that the queue buffer space should be no less than the bandwidth delay product. It was also proposed that delay should be the average end-to-end round trip time across all flows sharing the bottleneck link [47].

DT has a loss bias against bursty flows, because a bursty flow tends to have multiple packets arriving at the queue roughly at the same time. Therefore, the router will drop a bunch of packets from the same flow at the same time. DT also produces global synchronization, because the packets from all flows are dropped when the queue is full, leading all flows to back off at the same time. All the flows then follow the same algorithm of rate increase, leading to the same situation again, at roughly the same time [4]. Lockouts are another problem with DT. DT can allow a single connection to take over the bottleneck link thereby increasing unfairness, but not necessarily translating into lower link utilization [49].

Using RED [4, 22], a router will probabilistically drop an arriving packet even though the queue for the outbound interface is not full. The reason for this early drop comes from the fact that packet loss is the primary indicator of congestion for a TCP connection.

By dropping packets before a router's queue fills, the TCP connections sharing the queue will reduce their transmission rates and ensure that the queue does not overflow. Another consequence of the early dropping of packets is that RED enforces fairness, because the fraction of dropped packets for each connection is roughly proportional to the connection's share of the bandwidth.

The RED algorithm uses a weighted average of the total queue length to determine when to drop packets. When a packet arrives at the queue, if the weighted average queue length is less than a minimum threshold value (min_{th}), then no drop action will be taken and the packet will simply be enqueued. If the average is greater than the minimum threshold but less than a maximum threshold (max_{th}), an early drop test is performed as described below, up to the maximum drop rate (max_p), when the average queue size reaches max_{th}. An average queue length in the range between the thresholds indicates some congestion has begun and flows should be notified via packet drops. If the average is greater than the maximum threshold value, a forced drop operation will occur. An average queue length in this range indicates persistent congestion and packets must be dropped to avoid a persistently full queue. The forced drop is also used when the queue is full but the average queue length is still below the maximum threshold.

Using a weighted average, RED avoids overreacting to bursts and reacts to longer-term trends. Furthermore, because the thresholds are compared to the weighted average, it is possible that no forced drops will take place even when the instantaneous queue length is quite large. A graphical representation of the RED parameters is given in Figure 7.1.

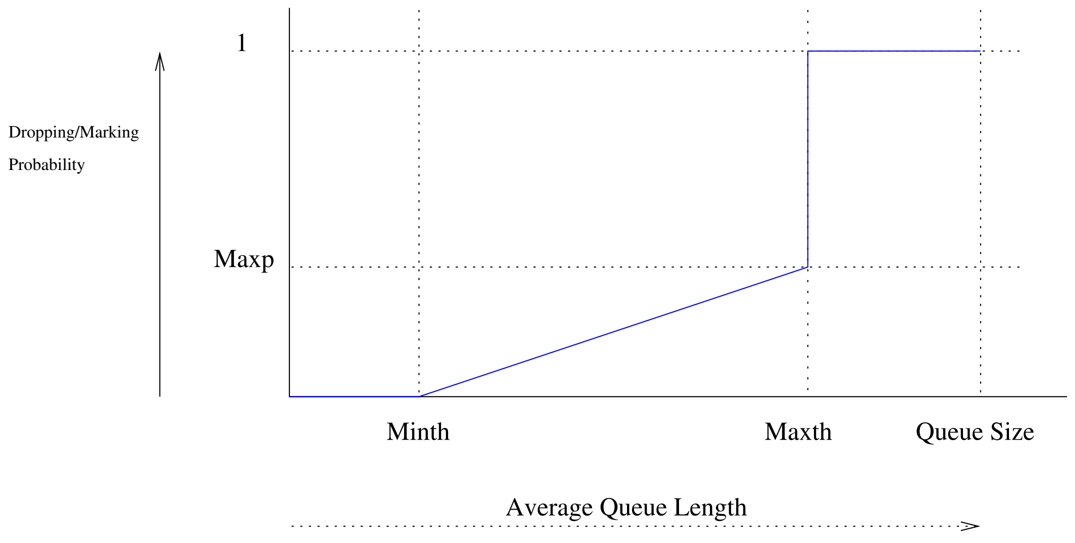


Figure 7.1: RED Parameters

Adaptive RED [20] is a variation of RED which retains RED's basic structure and dynamically adjusts the parameter max_p to keep the average queue size between the minimum and maximum thresholds. The objective of Adaptive RED is to reduce both packet loss rate and the variance in queuing delay.

RED's gentle mode [16] modifies RED's dropping function for the case when the average queue size exceeds max_{th}. The drop probability increases linearly between max_{th} and the buffer size with a slope of (1 - max_p) / max_{th}. Obviously, RED has to drop an arriving packet if the instantaneous queue size equals the total buffer size.

Another extension of RED is to mark the IP header instead of dropping packets, when the average queue size is between min_{th} and max_{th}, or between min_{th} and the buffer size (when adaptive RED and Gentle mode are used deployed together). Cooperating end systems should then use this IP mark as a signal that the network is congested and slow down the throughput. This is known as Explicit Congestion Notification (ECN) [53].

ECN aims to provide TCP with an alternative mechanism for detecting incipient congestion in the network. That is, a TCP sender that supports ECN does not have to solely depend on packet drops to detect congestion and limit its sending rate.

ECN requires support from the routers and the end hosts. Hosts negotiate ECN capability during the TCP connection setup. If both hosts are ECN-capable, the TCP sender indicates this by setting a bit in each outgoing packet. ECN-capable routers are responsible for monitoring congestion levels and marking packets of ECN-capable sources as congestion grows critical, instead of passively waiting until buffer space runs out and resorting to drops. ECN relies on the ability of the router to detect incipient congestion, unlike DT. Therefore, the router must use an Active Queue Management (AQM) mechanism, such as the one employed in RED.

Adaptive RED, with Gentle mode and support for ECN was used for all the simulations in this work that were conducted with RED router queuing management.

Impact of RED and DT in the Performance of HSTCP

The queue management scheme does not significantly affect the link utilization of HSTCP flows in the Ideal Condition. They presented similar results, as indicated earlier.

There is a considerable difference in the level of congestion event rate from RED to DT, as depicted in Figure 6.3(b). The use of RED causes a reduction in the number of the congestion events necessary to control the TCP sending rate. ECN plays an important role by notifying TCP senders of congestion build-up in a more effective manner than packet drops [49].

On the lossy link condition defined for this experiment there is no difference between the use of RED and DT, as mentioned before. The simple reason for this is that the router queue is not used at all, because the HSTCP flow set is limited by systemic loss instead of by congestion loss. So, the amount of traffic generated does not reach the link capacity. The queue management is only active for link loss rates set to lower than 10^{-5}, see Figure 6.8.

The impact of router queuing management is clear when HSTCP is submitted to bursty traffic. The link utilization for the set of HSTCP flows decreases slightly with RED, but it is not affected by perturbations when DT router queue policy is used. This can be seen in Figure 6.13(b). The constant congestion event rate for HSTCP presented in Figure 6.14(b) also illustrates this fact. RED decreases the bias against bursty traffic by increasing the congestion events of the non-bursty traffic, as depicted in Figure 6.14(a). This result is also found in the literature [3].

Also observing Figure 6.14(b) it is possible to realize that HSTCP flows seem to resist global synchronization better than REGTCP ones. The split in the results of congestion event rate for REGTCP flows seems to indicate that, at certain levels of perturbation, global synchronization can be triggered. Once it is started, the long-lived flows go to Slow-Start together, making it more likely that global synchronization is sustained by following perturbations, and the REGTCP flows do not recover.

Impact of the RED and DT on the Fairness

The general pattern of the aggregated relative fairness found when RED is used is also followed when DT router queue management is deployed. The difference is the higher amount of bandwidth that the HSCTP flows take from the REGTCP flows. In Ideal Condition, the ratio reaches values higher than 20 times for DT and has a high variability, as shown in Figure 6.6.

The use of DT as router queue management also impacts the amount of bandwidth stolen by the HSTCP flows from the REGTCP flows, as seen in Figure 6.7. Even though the amount of bandwidth stolen decreases as the number of flows increases, the distance between the amounts stolen when RED is used and when DT is used, increases slightly. This suggests that RED is doing a better job of avoiding unfair sharing among the flows in the router queue.

There are no relevant observations in Lossy Link Condition because the queue is not used in most of this experiment.

In Bursty Traffic conditions, the aggregated relative fairness is almost constant using DT, but seems to increase slightly when RED is used. This fact will be further explored when the HSTCP bulk data transfer capability is analyzed. Using DT, the HSTCP flows get between 10 and 15 times more bandwidth share than the REGTCP flows, as depicted in Figure 6.17.

The amount of bandwidth stolen by HSTCP flows from REGTCP flows also decreases with the increment of the number of perturbations, mainly because the effect bursty perturbations have on the REGTCP flows. But the amount of bandwidth stolen is still higher than with RED router queue management, as seen in Figure 6.18.

The experiment with Constant Link Loss Rate of 10^{-5} presents a new situation for the analysis of fairness. Figure 6.23(b) shows that the percentage of bandwidth used by both types of TCP is relatively invariant with the number of flows used, with DT router queue management. This is different from the result found when there was no link loss, as seen in Figure 6.4(b). The small variation that happens seems to indicate that the difference of link utilization between HSTCP flows and REGTCP flows becomes higher. This is clearly illustrated in Figure 6.24 and in Figure 6.25. Further investigation is necessary to explain this behavior.

Router queue management also makes a difference in the number of equivalent REGTCP flows for one HSTCP flow, when both are competing for the same link. Figure 6.19(b), the equivalence point for DT router queuing management (the point where 1 HSCTP flow is using the same bandwidth as N-1 REGTCP flows) is twice the value found when RED was used. The value found for DT under the conditions of this experiment was 13 REGTCP flows for 1 HSTCP flow. This figure highlights again the substantial impact that different types of queue management have on the behavior of TCP flows, and the consequent impact on the fairness observed when HSTCP and REGTCP flows are deployed together.

7.2.4 Can HighSpeed TCP be a substitute to other types of bulk data transfer?

Resource sharing is a strong driving force behind the development of computer communication networks. It permits a scarce and expensive resource to be used by geographically dispersed people. In certain types of communities, this resource could be required to carry large amounts of data produced by experiments, data collections, visualization tools, and so on. This scenario is particularly prevalent among scientific communities, such as high-energy physics, climate, astronomy and life sciences [34]. Large data sets produced at one site, sometimes must be analyzed at collaborating institutions around the world.

It is a massive problem to ensure that the data are distributed in an acceptable time for the computation in today's Internet. This problem has forced the design of techniques to overcome this challenge. One of the current techniques in place, to perform massive bulk data transfer, is the use of parallel TCP streams, as mentioned in the Background chapter. This technique is implemented by dividing the data to be transferred into N portions, and transferring each portion with a separate TCP connection. When running N connections, each parallel stream will be less likely to be selected to have its packets dropped, and therefore the aggregated amount of potential bandwidth which must go through premature congestion avoidance or Slow-Start is reduced.

It has been shown in [31] that the link utilization is proportional to the probability of loss and bandwidth delay product. The effect of N parallel streams is to reduce the bandwidth delay product experienced by a single stream by a factor of N, because they all share the same bandwidth.

Hacker and Athey [25] addressed how the use of parallel TCP connections increases aggregated throughput, and how to determine the number of TCP connections that are necessary to maximize throughput while avoiding network congestion. After developing a theoretical model and experiments, they concluded that the use of parallel TCP connections is equivalent to using a large MSS on a single connection, with the added benefit of reducing the negative effects of random packet loss. They also mentioned that the value for the number of parallel TCP connections should not be arbitrarily selected, because, if the selected value is too large, the aggregated flow may cause congestion and the throughput will not be maximized.

Further work of Hacker, Noble and Athey [26] proposed a fractional congestion control. In this model, a single TCP stream is told to increase its congestion window by only one packet for every N acknowledged packets, but decreases its window in the normal way. Fractional congestion control can be used to reduce the aggressiveness of parallel streams in the presence of congestion, but preserves much of their effectiveness in its absence. They proposed this control because when parallel flows compete with single stream, the former steal bandwidth from the latter.

The main advantages and disadvantages of using parallel streams are summarized and listed as follows:

ADVANTAGES:

- the Slow-Start is faster, because the aggregated flow grows N times faster;
- parallel streams can overcome the limitation on maximum TCP buffer sizes, as discussed in the Background section;
- its recovery is faster compared to a single TCP stream with large window, because the recovery of N individual streams is faster than one and, if only one stream in N experienced loss, the decrease will not be large for the aggregated flow.

DISADVANTAGES:

- parallel streams require special support in application programs, and consequently existent programs have to be changed;
- parallel streams may lose performance if the loss experienced by the aggregated flow is due to congestion;
- the selection of the number of parallel streams is problematic, because the network condition may change during a long data transmission, and a previously good condition could become bad later;

- parallel streams may be unfair to other TCP streams sharing the same network resources.

The deployment of HighSpeed TCP presents advantages and disadvantages compared to the use of parallel TCP streams technique:

ADVANTAGES:

- HighSpeed TCP does not require changes in application programs to use it, but instead a change in the TCP stack;
- its adaptability to varying loss rates better accommodates changes in the network conditions, even on a congested link (even though the use of fractional congestion control improves this point, it is dependent on the initial number of parallel streams set for the transmission);
- it is not necessary to know a priori the number of flows to transmit.

DISADVANTAGES:

- HighSpeed TCP has the same limitation as maximum TCP buffer size for a single TCP stream;
- HighSpeed TCP has just one control loop, instead of N in the case of parallel streams.

Both solutions have in common the potential unfairness to competing TCP transmission when they share a congested link, and both show a clear improvement in their transmission rates.

This section presents the theoretical performance of HSTCP compared to the parallel streams, as well as the expected fairness. After this, simulations comparing aspects of transfer rate and fairness on different network conditions are presented.

Theoretical performance and fairness of HighSpeed TCP and Parallel TCP Streams

If HSTCP is used on a network with unused bandwidth, and there are systemic packet losses, the packet losses experienced by a HSTCP stream will effectively determine the maximum throughput. The only possible modification to change this limit is to use different values for the HSTCP parameters (Low_Window, High_Window, and High_P). These parameters define the slope of the response function. After reaching the link capacity, the congestion packet losses limit further throughput increases. Parallel streams also have their throughput (and consequently aggregated throughput) determined by the packet loss rate experienced by each TCP stream. As the number of parallel streams increases, the packet loss perceived by each individual flow should be similar as long as only a few packets are queued in the routers. The performance of each scheme is presented in Figure 6.28. This graph presents only the performance when there are systemic packet losses and does not include losses due to congestion.

Observing their response functions, it is clear that a HSTCP flow may deliver a higher throughput when compared to a single REGTCP stream, or several parallel streams when very low systemic packet losses are present. Under low systemic loss rate, HSTCP is a strong candidate for bulk data transfer. For environments with high systemic packet losses, HSTCP's throughput is close to or equal to a REGTCP flow, and there is no particular advantage in terms of throughput.

If the link capacity is reached and the congestion loses dominates, no difference exist between the schemes (see Figure 6.27), except the fairness to other TCP flows. This is explored in the following discussion.

The relative fairness achieved by the use of parallel streams is directly related to the number of flows used (N), and independent of packet loss rate experienced by the flows [18, 25]. On the other hand, the relative fairness of HSTCP is a function of the congestion window size, and consequently a function of the congestion event rate, as can be seen in Figure 6.29. By this graph, it is possible to see the adaptability of just 1 HSTCP flow for different congestion event rates. The lower the packet loss rate, the higher the relative fairness.

One important aspect to observe is that both schemes are not hurting other TCP flows while systemic losses are dominant since there is still bandwidth available.

When *congestive* packet losses begin to emerge, the link capacity is reached, and a new dynamic for fairness appears. After this point, the fairness will be determined by the router queue management policy, traffic volume, parameters of each scheme (N for the number of parallel streams and "Low_Window, High_Window, and High_P" for HSTCP). After this turning point, the parallel streams begin to steal bandwidth from the other TCP flows competing for the bandwidth. The amount stolen is proportional to the number of parallel flows deployed and to the volume of the concurrent traffic.

When a HSTCP flow is deployed, the amount of bandwidth stolen is function of its parameters and also to the concurrent traffic. The amount of bandwidth stolen decreases as the competing traffic increases. The influence of router queue management is expressed in Figure 6.19(a) and in Figure 6.19(b). These graphs show that to use the same amount of bandwidth, requires a greater number of REGTCP flows when DT is deployed than the number of REGTCP flows necessary when RED is used.

It should be noticed that a HSTCP flow loses more of its bandwidth share than a set of parallel streams, when there is a variation in the competing traffic. This is expressed in the slope of the lines on Figure 6.28. As the competing traffic changes, so does the general congestion event rate.

This is an important point to consider when a bulk data transfer is performed in a congested link. The greater variation presented by HSTCP compared to parallel streams permits it to adapt quickly to variation in traffic and congestion event rate.

As discussed previously, there are no clear disadvantages to using HSTCP when compared to parallel streams for bulk data transfer. Even the limitation of maximum TCP buffer size may be a common problem if a small number of parallel TCP streams are use. The possibility of changing only the TCP stack instead of changing programs already in use is very attractive. Fairness is a common concern for both schemes, however our opinion is that HSTCP presents a better adaptability to an environment of a variable congestion event rate. Parallel streams may also present better adaptability (using fractional congestion control or some other type of adaptive control), but at a cost of their simplicity.

We present, in the following paragraphs, the experiments developed to observe the aspects of performance and fairness of both bulk data transfer schemes.

Simulation comparison of schemes for bulk data transfer

Two experiments were developed to compare the deployment of parallel streams and HSTCP. The first one deals with reactions to a lossy link and the second shows the behavior when bursty traffic is present. In both experiments 10 REGTCP flows are present to measure the impact of both bulk data transfer schemes on other long lived flows.

PARALLEL STREAMS ON LOSSY LINK CONDITION

The performance of parallel streams in an environment with systemic packet losses is defined by the congestion event rate and the number of parallel streams, as said before, and confirmed on Figure 6.31(a). After the bandwidth limit is reached, the link utilization stays constant. The same behavior is observed for 1 HSTCP flow on the same graph. For loss rates higher than 10^{-3}, both schemes perform badly. This poor performance is the result of a large increase in the congestion event rate, mainly due to packet retransmissions, as depicted in Figure 6.32(a).

The impact of the use of parallel streams and HSTCP on other long-lived flows is presented on Figure 6.30(a). This graph shows the aggregate link utilization of 10 REGTCP flows when both schemes for bulk data transfer are deployed and also when there is no other interference. No difference exists in performance for these 10 flows with and without competition before the total link bandwidth is reached, both set of traffic (10 REGTCP long-lived flows and the competing traffic from bulk data transfer) have room to grow. After the bandwidth is fully utilized, there is no variation in the amount of bandwidth used by the 10 REGTCP flows.

Figure 6.33(a) shows how many times more bandwidth a set of parallel streams is using than a single REGTCP flow (one tenth of the aggregate link utilization of all 10 REGTCP flows). It is clear that this ratio is roughly constant over a wide range of link loss rates. This behavior only changes when there is a heavy packet loss rate, higher than 10^{-4}. In comparison, the ratio of relative fairness, when HSTCP is used, is not constant. It changes with the level of link loss rate, because HSTCP was designed to perform in this way. This changing behavior reflects its adaptability to a changing link loss environment. It can represent one single REGTCP flow when link loss rate is about 10^{-3} and 5 REGTCP flows when the link loss is around 10^{-5}. This adaptability represents an advantage over the use of parallel streams, because it is hard to know a priori which will be the minimum systemic loss rate and maximum bandwidth available, in order to avoid hurting too much the other TCP flows competing for the bandwidth.

When DT router queue management is used, the difference of performance compared to RED only happens after the bandwidth limit is reached. Also, the link utilization is kept similar for parallel streams after bandwidth limit is reached, Figure 6.30(b).

DT permits a higher aggressiveness from HSTCP. HSTCP takes more bandwidth from the 10 REGTCP flows, as seen in Figure 6.30(b), and consequently presents a wide range of relative fairness, as shown in Figure 6.33(b).

PARALLEL STREAMS ON BURSTY TRAFFIC CONDITION

The performance of bulk data transfer methods is stressed when they run in a environment with bursty traffic. The performance of parallel streams tends to decrease as the number of perturbations increase, as seen on Figure 6.35(a). This behavior is the same presented by other flows using REGTCP when submitted to similar situations, as shown in Figure 6.13(a).

The performance of 1 HSTCP flow is much less sensitive to this environment. Indeed, the performance of HSTCP, when competing against 10 REGTCP flows improves as the number of perturbations increases. This is clear on Figure 6.35(a) and Figure 6.36(a). The first thought could be that HSTCP is stealing bandwidth from REGTCP flows, but Figure 6.37(a) shows that the amount stolen decreases with an increase in the number of bursty flows. The only possible explanation for this behavior is that HSTCP is using the bandwidth share left by REGTCP flows, because they were hurt by the bursty traffic. This presents an excellent feature of HSTCP in this environment (bursty traffic and RED router queue policy) when compared with parallel streams: HSTCP is able to use the bandwidth share left by REGTCP flows when submitted to bursty traffic, and parallel streams are unable to do this.

For the parallel streams method, the relative fairness is kept almost constant over the range of a number of perturbations and is proportional to the number of parallel TCP streams. As said above, for the case when HSTCP is present, the relative fairness increases as the number of perturbations increase, and the HSTCP flow is able to use more bandwidth. These facts are shown on Figure 6.36(a). As pointed out before, it is clear that HSTCP has a better adaptability, and uses more bandwidth, without significantly harming the other flows.

The situation changes when DT is used as router queuing management. The reduced performance of parallel streams for a large number of perturbations is not clear, at least for the number of perturbations used on this experiment, Figure 6.35(b). So, the performance of parallel streams is kept almost constant as the number of perturbations increases. The performance of HSTCP follows the same behavior. The only difference from RED, is that HSTCP uses much more bandwidth.

The relative fairness remains unchanged for parallel streams, as it was with the use of RED. For HSTCP, the relative fairness could spread over a wide range of values, as seen on Figure 6.36(a). Even though a certain degree of uncertainty is present in the relative fairness, the link utilization of HSTCP is relatively constant, Figure 6.35(b), and the amount of bandwidth stolen from the REGTCP flows competing against the HSTCP flow, decreases when the number of perturbations is high, as seen in Figure 6.37(b).

7.3 Other issues

This section explores some issues observed during the development of this work. Such issues are not directly related to the main investigation, but have some impact in the results.

7.3.1 Slow-Start problem

The standard Slow-Start algorithm provides an exponential increase in the size of the congestion window, by doubling its size each time an ACK packet is received. The reason for the fast growth is to quickly probe the network capacity, and begin quickly to transmit near to the link capacity. However, doubling the congestion window each RTT, on a high capacity link, can easily result in thousands of packets being dropped in one round-trip time. This drop of a large number of packets can result in unnecessary retransmit timeouts for the TCP connection. The TCP connection could end up in a Congestion Avoidance phase with a very small congestion window, and could take a large number of round-trip times to recover its old congestion window [17]. Therefore, the traditional Slow-Start algorithm has a poor performance for high congestion windows.

This problem could seriously compromise the performance results of this present work. This behavior happens with HSTCP and REGTCP since both use the same Slow-Start algorithm. Sometimes this affects the performance of the flows also in the steady-state, preventing flows to utilize more bandwidth.

This problem was identified during the experiments described in this work and the Draft "Limited Slow-Start for TCP with Large Congestion Windows" [17] was proposed as a solution. Limited Slow-Start was used in this study.

The parameter MAX_SSTHRESH was introduced, and the Slow-Start was modified for large values of the congestion window, as mentioned in Section 2.3

It is necessary to tune the MAX_SSTHRESH parameter to have reasonable performance in Slow-Start and to avoid having flows stalled in the beginning of their transmission. The proposed MAX_SSTHRESH of 100 packets [17] was used in the present study.

The effect of massive packet loss can be seen in Figure 6.39. This graph shows the effect of the losses on the sequence numbers of a HSTCP flow when using different parameters and algorithms for Slow-Start.

Table 6.1 and Figure 6.38 give a comparison among various Slow-Start Conditions. The MAX_SSTHRESH = 0 represents the standard algorithm. It is clear that even though a flow with standard algorithm reaches the congestion avoidance phase faster then the others, it will have a large packet loss.

7.3.2 Neighborhood of bandwidth limit

Figure 3.1 shows the theoretical response function of HSTCP, when it is submitted to different congestion event rates. It also shows the link capacity used in this work. The intersection between these two lines defines the maximum theoretical performance of 1 HSTCP flow in this study.

In this work it was possible to see that 1 HSTCP flow may be close to this point. From data collected in the experiment of Figure 6.1(a), it was possible to say that a flow had 1 congestion event for about every 638,000 packets sent, in approximately 7.73 seconds. It resulted in a congestion event rate of $1.56*10^{-6}$ and a transmission rate of 8,244 Pkts/RTT, close to the bandwidth limit of 8,333 Pkts/RTT, or 98.9% of the link bandwidth.

7.3.3 Problem in the implementation of TCP SACK for Large Congestion Windows

HighSpeed TCP is designed to run in a very large window regime. This condition is intended to be reached in future networks. For this reason, implementations of the TCP algorithm have hardly been tested in this condition, and no experience has been developed.

During the development of this work, a permanent performance loss was observed during the deployment of HSTCP flows, particularly when a packet loss happened with a large congestion window. The symptom was a double cut in the congestion window, as seen in Figure 7.2.

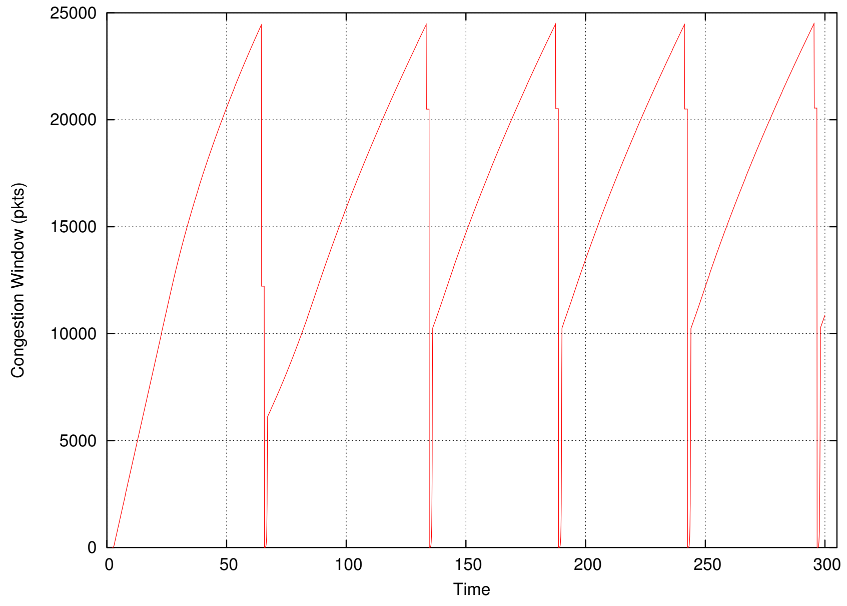


Figure 7.2: Evolution of Congestion Window - Buggy HSTCP - DT

After a long investigation, it was noticed that the implementation of TCP SACK in the NS-2 simulator, before version ns2-1b9a (tcp.cc 1.52, tcp-sink.cc 1.46, scoreboard.cc 1.14), did not correctly report SACK blocks for windows greater than 1000 packets.

Even though this bug was fixed, a scalable solution has not been implemented. The implementation works according to the respective RFC, but, for high speed TCP connections where CWND is higher than 10,000 packets, the simulation slowed down dramatically.

This problem points out the increasing difficult of having a working TCP implementation for high speed networks. The algorithm used in a previous implementation may not fit well in a high performance scenario. The second aspect highlighted is that certain implementation problems are only noticeable in specific scenarios, and the complexity of implementing multiple algorithms for a single TCP is a challenge of design and testing.

Chapter 8

Conclusion and Future Work

The pressure for more network bandwidth has produced technologies that have increased the bandwidth capacity available several times. New technologies, such as optical links have opened the opportunity to transfer several terabytes in a relatively small time. However, standard TCP has a difficult time reaching full utilization of optical links, particularly in wide-area connections. So, many network applications are unable to take full advantage of these new high-speed networks, and utilize the available bandwidth.

The purpose of this work was to study the deployment of the recent proposal of HighSpeed TCP in high speed long distance links. We worked around four main points as follows:

- the behavior of HighSpeed TCP in situations where Standard TCP has weak performance
- the possibility of using HighSpeed TCP together with Standard TCP and maintaining fairness
- the effect of the router queuing policy on the performance of HighSpeed TCP and on the fairness between HighSpeed TCP and Standard TCP
- the possibility of using HighSpeed TCP as a substitute for the other existing solutions for bulk data transfer

As result of this work, we found that HighSpeed TCP indeed performs better than Standard TCP for high-speed long-distance links. HighSpeed TCP increases its throughput faster and its recovery from a congestion event takes less time. This characteristic increases its average link utilization.

We also showed that the bandwidth share used by the HighSpeed TCP flows was higher than that used by Standard TCP flows, when both types of flows competed for the same link. However, it was noticeable that the proportion of the bandwidth used by the HighSpeed TCP flows decreased as the total number of congestion events increased.

The change of the queue management scheme did not significantly affect the link utilization of HighSpeed TCP flows in most cases. The general pattern of the relative fairness found, when RED was used, was also followed when DT router queue management was deployed. The difference was the higher amount of bandwidth that HighSpeed TCP flows took from Standard TCP flows, when DT was used.

We saw that HighSpeed TCP only requires changes in the TCP stack in the sender to be used. This represents an advantage over other types of bulk data transfer such as parallel streams, when it is necessary to change the programs and know a priori the number of parallel flows to use. Fairness is a concern for the deployment of HighSpeed TCP, however our opinion is that HighSpeed TCP presents better adaptability to an environment of variable congestion event rates.

Some steps should follow this work. The most important now is to observe the behavior of HighSpeed TCP in a real network with high-speed and long-delay links. Some tests had already begun [9].

We foresee that a study of the parameters of HighSpeed TCP would be necessary to explore different combinations of parameters settings. A change in the parameters will change the response function for different levels of congestion event rate. The aggressiveness and relative fairness compared to Standard TCP will be affected. Other transfer functions, beyond the linear one, could also be tested.

The behavior in transients is another area to be investigated. The reaction of HighSpeed TCP to a sudden change in the bandwidth available is important to see its recovery and stabilization time. Part of this examination was done, but there is room for further investigation.

One question to be explored is whether modifying TCP congestion algorithms, like HighSpeed TCP, to increase the transmission throughput is better than simply modifying the standard frame size for the Ethernet. Nowadays the Ethernet frame size is frozen at 1500 bytes. This size represents the majority of packet sizes present in today's Internet. The present Ethernet frame size restricts throughput in high-speed long-delay links.

The final point to think is about a review in the concept of fairness as used nowadays for "TCP-friendliness". How is possible to ask that new TCP implementation be completely fair to classical TCP versions if it is know that these versions perform poorly in high bandwidth delay product links? In this case, keep fairness also means to maintain a poor performance. Clearly, this is not acceptable.

The real benefits of increasing availability of long distance gigabit links will not be completely realized for high bandwidth demanding applications, such as bulk-data transfer, multimedia web streaming and computational grids for high-performance computing, if the main transport protocol does not succeed in using the available link capacity.

Bibliography

[1] M. Allman, V. Paxson, and R. Stevens. TCP congestion control. Internet Engineering Task Force, April 1999. RFC2581.

[2] J. Bolot. Characterizing end-to-end packet delay and loss in the internet. *Journal of High Speed Networks*, 2(3):289–298, September 1993.

[3] T. Bonald, M. May, and J. Bolot. Analytic evaluation of RED performance. In *Proceedings of the 2000 IEEE Computer and Communications Societies Conference on Computer Communications*, pages 1415–1424, August 2000.

[4] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshall, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, and L. Zhang. Recommendations on queue management and congestion avoidance in the Internet. Internet Engineering Task Force, April 1998. RFC2309.

[5] R. Braden. Requirements for internet hosts - communication layers. Internet Engineering Task Force, October 1989. RFC1122.

[6] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. Advances in network simulation. *IEEE Computer*, 33(5):59–67, May 2000.

[7] R. Carlson. Tackling the end-to-end performance problem. Large Scale Networking Workshop, March 2001. URL http://www.ngi-supernet.org/lsn2000/Argonne_Natl_Lab-Carlson.pdf.

[8] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Journal of Computer Networks and ISDN Systems*, 17(1):1–14, June 1989.

[9] F. Coccetti and L. Cottrell. TCP stacks comparison with a single stream. Stanford Linear Accelerator Center, March 2003. URL http://www-iepm.slac.stanford.edu/monitoring/bulk/fast/tcp-comparison.html.

[10] Euclidian Consulting. Description and resoluation to reported packet loss problem with toshiba pcx1000 cable modems. Avaliable at http://www.cablemodemhelp.com/pcx1000.htm.

[11] T. Duningan, M. Mathis, and B. Tierney. A TCP tuning daemon. In *Procedings of IEEE Super Computing 2002*, 2002.

[12] P. Dykstra. Gigabit ethernet jumbo frames, December 1999. URL http://sd.wareonearth.com/˜phil/jumbo.html.

[13] K. Fall and S. Floyd. Simulation-based comparisons of Tahoe, Reno and SACK TCP. *Computer Communication Review*, 26(3):5–21, 1996.

[14] M. Fisk and W. Feng. Dynamic right-sizing in TCP. In *Procedings of Los Alamos Computer Science Institute Symposium*, October 2001.

[15] S. Floyd. Congestion control principles. Internet Engineering Task Force, September 2000. RFC2914.

[16] S. Floyd. Recommendation on using the gentle variant of RED, March 2000. URL http://www.icir.org/floyd/red/gentle.html.

[17] S. Floyd. Limited slow-start for TCP with large congestion windows, May 2002. Internet draft draft-floyd-tcp-slowstart-00b.txt, work in progress.

[18] S. Floyd. Highspeed TCP for large congestion window, February 2003. Internet Draft draft-floyd-tcp-highspeed-01.txt, work in progress.

[19] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Transactions on Networking*, 7(4):458–472, August 1999.

[20] S. Floyd, R. Gummadi, and S. Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management. URL http://www.icir.org/floyd/papers/adaptativeRed.pdf, August 2001.

[21] S. Floyd and V. Jacobson. On traffic phase effects in packet-switched gateways. *Internetworking: Research and Experience*, 3(3):115–156, September 1992.

[22] S. Floyd and V. Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[23] S. Floyd, S. Ratnasamy, and S. Shenker. Modifying TCP's congestion control for high speeds. Preliminary Draft. URL http://www.icir.org/floyd/papers/hstcp.pdf, 2002.

[24] S. Gadde, J. S. Chase, and A. M. Vahdat. Coarse-grained network simulation for wide-area distributed systems. In *Communication Networks and Distributed Systems Modeling and Simulation Conference*, January 2002.

[25] T. Hacker, B. Athey, and B. Noble. The end-to-end performance effects of parallel TCP sockets on a lossy wide-area network. In *16th International Parallel and Distributed Processing Symposium*, April 2002. URL http://www.cnaf.infn.it/ferrari/papers/tcp/IPDPS.pdf.

[26] T. J. Hacker, B. D. Noble, and B. D. Athey. The effects of systemic packet loss on aggregate TCP flows. In *IEEE/ACM Supercomputing 2002: High Performance Networking and Computing*, November 2002.

[27] G. Hasegawa and M. Murata. Survey on fairness issues in TCP congestion control mechanisms. *IEICE Transactions on Communications*, E84-B(6):1461–1472, June 2001.

[28] W. Huntoon, T. Dunigan, and B. Tierney. The Net100 project: Development of network-aware operating systems. URL http://www.net100.org.

[29] Linux Headquarters Inc. Linux kernel 2.4 variables, 2002. URL http://www.linuxhq.com/kernel/v2.4/doc/networking/ip-sysctl.txt.html.

[30] B. Irwin and M. Mathis. Web100: Facilitating high-performance network use, January 2001. URL http://www.internet2.edu/E2E/papers/20010109-E2EPM-Irwin.pdf.

[31] V. Jacobson. Congestion avoidance and control. In *Proceedings of the ACM SIGCOMM '88 Conference on Communications Architectures and Protocols*, volume 18, pages 314–329, Stanford, CA, August 1988.

[32] V. Jacobson, R. Braden, and D. Borman. TCP extensions for high performance. Internet Engineering Task Force, May 1992. RFC1323.

[33] C. Jinand, D. Wei, S. H. Low, G. Buhrmaster, J. Bunn, D. H. Choe, R. L. A. Cottrell, J. C. Doyle, W. Feng, O. Martin, H. Newman, F. Paganini, S. Ravot, and S. Singh. FAST TCP: From theory to experiments. Submitted to IEEE Comunications Magazine, Internet Technology Series, April 2003.

[34] W. Johnston, W. Kramer, J. Leighton, and C. Catlett. A vision for DOE scientific networking driven by high impact science, March 2002. URL http://www.lbl.gov/CS/html/Network_Vision_Whitepaper.pdf.

[35] D. Katabi, M. Handley, and C. Rohrs. Internet congestion control for high bandwidth-delay product networks. In *ACM Sigcomm 2002*, August 2002.

[36] J. Kulik, R. Coulter, D. Rockwell, and C. Partridge. Paced TCP for high delay-bandwidth networks. In *Proceedings of IEEE Globecom*, December 1999.

[37] K. Kurata, G. Hasegawa, and M. Murata. Fairness comparisons between TCP Reno and TCP Vegas for future deployment of TCP Vegas. In *Proceedings of IEEE INET 2000*, July 2000.

[38] T. Lakshman and U. Madhow. Performance analysis of window-based flow control using TCP/IP: Effect of high bandwidth-delay products and random loss. In *Fifth International Conference on High Performance Networking*, pages 135–149, June 1994.

[39] J. Lee, D. Gunter, B. Tierney, W. Allock, J. Bester, J. Bresnahan, and S. Teck. Applied techniques for high bandwidth data transfers across wide area network. In *Computing in High Energy and Nuclear Physics*, Beijing, China, April 2001. LBNL-46269.

[40] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic (extended version). In *IEEE/ACM Transactions on Networking*, volume 2, pages 1–15, 1994.

[41] S. Low, L. Peterson, and L. Wan. Understanding TCP Vegas: A duality model. In *Proceedings of ACM SIGMETRICS 2001*, Cambridge, USA, June 2001.

[42] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP selective acknowledgment options. Internet Engineering Task Force, October 1996. RFC2018.

[43] M. Mathis, J. Semke, J. Mahdavi, and T Ott. The macroscopic behavior of the TCP congestion avoidance algorithm. *Computer Communications Review*, 27(3), July 1997.

[44] S. McCreary and K. Claffy. Trends in wide area ip traffic patterns - a view from ames internet exchange. In *13th ITC Specialist Seminar on Internet Traffic Measurement and Modelling*, Monterey, CA, September 2000.

[45] J. Nagle. Congestion control in IP/TCP Internetworks. Internet Engineering Task Force, January 1984. RFC896.

[46] R. Nitzan and B. Tierney. Experiences with TCP/IP over an ATM OC12 WAN. Technical Report LBNL-44765, Lawrence Berkeley National Laboratory, April 1999.

[47] C. Partridge. Buffer size = bw-delay product, part 2. End-to-End Mail List Archive, April 1998. URL ftp://ftp.isi.edu/end2end/end2end-interest-1998.mail.

[48] V. Paxson. *Measurements and Analysis of End-to-end Internet Dynamics*. PhD thesis, University of California at Berkeley, April 1997.

[49] K. Pentikousis, H. Badr, and B. Kharma. On the performance gains of TCP with ECN. In *Proceedings of the 2nd European Conference on Universal Multiservice Networks*, Colmar, France, April 2002.

[50] The VINT Project. NS-2 network simulator. URL http://www.isi.edu/nsnam/ns.

[51] K. Ramakrishnan and S. Floyd. A proposal to add explicit congestion notification (ECN) to IP. Internet Engineering Task Force, January 1999. RFC2481.

[52] K. Ramakrishnan, S. Floyd, and D. Black. The addition of explicit congestion notification (ECN) to IP. Internet Engineering Task Force, September 2001. RFC3168.

[53] J. Salim and U. Ahmed. Performance evaluation of explicit congestion notification (ECN) in IP networks. Internet Engineering Task Force, July 2000. RFC2884.

[54] J. Semke, J. Mahdavi, and M. Mathis. Automatic TCP buffer tuning. *Computer Communication Review*, 28(4):315–323, October 1998.

[55] M. Sooriyabandara and G. Fairhurst. Performance limitations due to TCP burstiness in GEO satellite networks with limited buffering. In *London Communications Symposium*, pages 127–130, London, UK, September 2000.

[56] R. Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[57] J. Stone and C. Partridge. When the CRC and TCP checksum disagre. In *ACM SIGCOMM Computer Communication Review: Proc. of the conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 309–319, Stockholm, Sweden, August 2000.

[58] B. Tierney. TCP tuning guide for distributed aplications on wide area networks. Technical Report LBNL-45261, Lawrence Berbeley National Laboratory, February 2001.

[59] B. Welch. *Practical Programming in Tcl and Tk*. Prentice-Hall Inc., Upper Sadle River, New Jersey, USA, 2000.

[60] B. White, J. Lepreau, L. Stoller, R. Ricci, S. G. M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Fifth Symposium on Operating System Design and Implementation*, 2002.

[61] J. Winder. Equation-based congestion control. Master's thesis, University of Mannheim, February 2000.

[62] Y. Yang and S. Lam. General AIMD congestion control. Technical Report TR-200009, Department of Computer Science, University of Texas at Austin, May 2000. URL http://www.cs.utexas.edu/users/lam/NRL/TechReports.

[63] L. Zhang, S. Shenker, and D. Clark. Observations on the dynamics of a congestion control algorithm: The effects of two way traffic. In *Proceedings of ACM SIGCOMM'91*, pages 133–148, Zurich, Switzerland, September 1991.

Appendix A

Simulation Scripts

A.1 behavior.tcl

```
########################################
# General HSTCP Behavior - behavior.tcl #
# Evandro de Souza - Apr/2002           #
# LBNL/Unicamp/Embrapa                  #
########################################

#
# Detailed tracing of HSTCP behavior under different conditions
#

#----- Input Parameters -----#
# Bottleneck link bandwidth (in MBits/s)
set bandwidth [lindex $argv 0]
# Bottleneck link delay (in miliseconds)
set delay [lindex $argv 1]
# Queue Type (0 = RED; 1 = DROPTAIL)
set queue [lindex $argv 2]
# Simulation time (in seconds)
set endtime [lindex $argv 3]
# Number of hstcp flows
set hstcpflows [lindex $argv 4]
# Number of regular tcp flows
set regtcpflows [lindex $argv 5]
# Number of web sessions
set web_sessions [lindex $argv 6]
# HSTCP low_window (in packets)
set low_window [lindex $argv 7]
# HSTCP high_window (in packets)
set high_window [lindex $argv 8]
# HSTCP high packet loss
set high_p [lindex $argv 9]
# HSTCP high decrease
set high_decrease [lindex $argv 10]
# Simulation with HSTCP ($hstcp == 1) or without HSTCP ($hstcp == 0)
set hstcp [lindex $argv 11]
# Queue Monitor (qflag == 1) --> present; (qflag == 0) --> absent
set qflag [lindex $argv 12]
# Error Model Loss Rate
set em_rate [lindex $argv 13]
```

```
# Number of Perturbation Flows
set nrpertflows [lindex $argv 14]
# Number of UDP Pareto Flows - Traffic Generation
set nrudpflows [lindex $argv 15]
# Time of ON and time OFF for UDP Pareto flows in miliseconds
set onofftime [lindex $argv 16]
# Switch to disable noise traffic
set noisetraf [lindex $argv 17]


#
#----- Variables for final aggregate report -----#
#
# Total link utilization in the 1st half simulation time
set tlu_1 0
# Total link utilization in the whole simulation time
set tlu_t 0
# Aggregate hstcp link utilization in the 1st half of simulation time
set agghs_1 0
# Aggregate hstcp link utilization in the whole simulation time
set agghs_t 0
# Aggregate regtcp link utilization in the 1st half of simulation time
set aggreg_1 0
# Aggregate regtcp link utilization in the whole simulation time
set aggreg_t 0
# Aggregate perttcp link utilization in the 1st half of simulation time
set aggpert_1 0
# Aggregate perttcp link utilization in the whole simulation time
set aggpert_t 0

# Average hstcp link utilization in the 1st half of simulation time
set avghs_1 0
# Average hstcp link utilization in the whole simulation time
set avghs_t 0
# Average regtcp link utilization in the 1st half of simulation time
set avgreg_1 0
# Average regtcp link utilization in the whole simulation time
set avgreg_t 0
# Drops in the 1st half of simulation time
set drops_1 0
# Marks in the 1st half of simulation time
set marks_1 0
# Drops in the whole simulation time
set drops_t 0
# Marks in the whole simulation time
set marks_t 0
# Packets sent in the 1st half of simulation time
set packets_1 0
# Packets sent in the whole of simulation time
set packets_t 0

# Aggregated HSTCP dropped packets in the 1st half of simulation
set agghsdp_1 0
# Aggregated HSTCP dropped packets in the whole simulation
set agghsdp_t 0
# Aggregated HSTCP marked packets in the 1st half of simulation
set agghsmk_1 0
# Aggregated HSTCP marked packets in the whole simulation
set agghsmk_t 0
# Aggregated REGTCP dropped packets in the 1st half of simulation
set aggregdp_1 0
# Aggregated REGTCP dropped packets in the whole simulation
set aggregdp_t 0
# Aggregated REGTCP marked packets in the 1st half of simulation
set aggregmk_1 0
```

```
# Aggregated REGTCP marked packets in the whole simulation
set aggregmk_t 0

#
#----- Constant Parameters -----#
#
# maximum number of nodes on each side of bottleneck link
set max_nrnodes 25
# number of small tcp flows
set smalltcpflows 10
# TCP packet size less header (1460, 8960, 536)
set pktsize 1460


#----- Set Packet Format -----#
remove-all-packet-headers       ; # removes all except common
add-packet-header Flags IP TCP  ; # hdrs reqd for TCP


#----- Load auxiliary files and procedures-----#
source utils.tcl
source web.tcl


#----- Print information at console about class of the traffic -----#
proc printlegend {} {
global  bandwidth delay queue endtime hstcpflows regtcpflows web_sessions low_window
high_window high_p high_decrease hstcp qflag frep em_rate nrpertflows

  # write to file
  puts $frep "behavior.tcl $bandwidth $delay $queue $endtime $hstcpflows $regtcpflows
$web_sessions $low_window $high_window $high_p $high_decrease $hstcp $qflag $em_rate
$nrpertflows"
  puts $frep " "
  puts $frep "forward HSTCP traffic:            flows 0  to [expr $hstcpflows - 1]"
  puts $frep "forward Regular TCP traffic:      flows 50 to [expr 50 + $regtcpflows - 1]"
  puts $frep "forward WEB traffic:              class 100"
  puts $frep "backward WEB traffic:             class 120"
  puts $frep "forward small TCP traffic:        class 140"
  puts $frep "backward small TCP traffic:       class 160"
  puts $frep "forward perturbation TCP traffic: class 300"

  # print to console
  puts "behavior.tcl $bandwidth $delay $queue $endtime $hstcpflows $regtcpflows
$web_sessions $low_window $high_window $high_p $high_decrease $hstcp $qflag
$em_rate $nrpertflows"
  puts " "
  puts "forward HSTCP traffic:            flows 0  to [expr $hstcpflows - 1]"
  puts "forward Regular TCP traffic:      flows 50 to [expr 50 + $regtcpflows - 1]"
  puts "forward WEB traffic:              class 100"
  puts "backward WEB traffic:             class 120"
  puts "forward small TCP traffic:        class 140"
  puts "backward small TCP traffic:       class 160"
  puts "forward perturbation TCP traffic: class 300"

}


#----- Set simulation handle -----#
set ns [new Simulator]

# set tf [open simul.tr w]
# $ns trace-all $tf

# set namtf [open simul.nam w]
# $ns namtrace-all $namtf


#----- Set bottleneck link nodes -----#
set n1 [$ns node]
set n2 [$ns node]


#----- Initialize special parameters for RED queue (manual item 7.3) -----#
#
# increase the pkt drop rate slowly from max_p to 1 as the avg queue size ranges from
# maxthresh to twice maxthresh (../tcl/test/test-suite-red.tcl)
Queue/RED set gentle_ true
# ? set on adaptative RED ?
Queue/RED set adaptive_ 1
# minimum for max_p (../queue/red.cc) (change Sally suggestion in Oct/16/2002)
#Queue/RED set bottom_ 0.001
Queue/RED set bottom_ 0.0001
# mininum threshold for the average queue size in packets
Queue/RED set thresh_ 0
# maximum threshold for the average queue size in packets
Queue/RED set maxthresh_ 0
# used in exponential-weigthed moving avg for calculating the avg queue size
Queue/RED set q_weight_ 0
# if true use DT rather than random drop when queue overflows the avg queue size
Queue/RED set drop_tail_ 0
# if true, mark pkts by setting the congestion indication bit in pkt header
Queue/RED set setbit_ 1
# percentage of packet time constant - max umber of (avg sized) pkets per seconod which
# can can be placed on the link
Queue/RED set targetdelay_ 0.005


#----- Set parameters for TCP conections (manual item 30.1.4)-----#
# TCP react to ecn bit 1=yes
Agent/TCP set ecn_ 1
# maximum bound on window size (large enough to not impose limits)
Agent/TCP set window_ 100000
# packet size used by sender
#Agent/TCP set packetSize_ 1460
Agent/TCP set packetSize_ $pktsize
# !=0 adds random time between sends
Agent/TCP set overhead_ 0.000008
# Change sshthresh in avoid loss packets in the slow-start
Agent/TCP set max_ssthresh_ 100


#----- Set bottleneck link -----#
if {$queue == 0} {
  $ns duplex-link $n1 $n2 [expr $bandwidth]Mb [expr $delay]ms RED
} else {
  $ns duplex-link $n1 $n2 [expr $bandwidth]Mb [expr $delay]ms DropTail
}
# queue-limit in packages
 $ns queue-limit $n1 $n2 [expr floor((2 * $delay * 0.001 * $bandwidth * 1000000 )/
                                    (8 *  1500)) ]
 $ns queue-limit $n2 $n1 [expr floor((2 * $delay * 0.001 * $bandwidth * 1000000 )/
                                    (8 *  1500)) ]

# puts "Queue Size: [expr floor((2 * $delay * 0.001 * $bandwidth * 1000000 )/(8 *  1500)) ]"


#----- Set Error Model (manual 13.4) -----#
# Only do the error model if error rate greater than zero
if {$em_rate > 0} {
  set em [new ErrorModel]
  # set unit of drops
  $em unit pkt
  # set the error rate
  $em set rate_ $em_rate
  # set the random variable
  $em ranvar [new RandomVariable/Uniform]
  # set target for dropped packets
  $em drop-target [new Agent/Null]
  # set the link to apply this model
  $ns lossmodel $em $n1 $n2
}


#----- Set Queue MONitor and attach to bottleneck link nodes -----#
#----- Set Flow MONitor and attach to the bottleneck link -----#
# qmon = queue monitor handler
# fmon = flow monitor handler
if {$qflag == 1} {
  set qmon [$ns monitor-queue $n1 $n2 ""]
  print-qstats 0.0 $endtime queue
}
set fmon [$ns makeflowmon Fid]
$ns attach-fmon [$ns link $n1 $n2] $fmon


#----- Set Randon Number Generator (manual item 21.1) -----#
# seeds the RNG heuristically
set rng [new RNG]
#$rng seed 0


#----- Create network topology (nodes and links) each link with different delay-----#
# node_(s$i) = source nodes
# node_(k$i) = sink nodes
for {set i 0} {$i < $max_nrnodes} {incr i} {
  set node_(s$i) [$ns node]
  set node_(k$i) [$ns node]

  $ns duplex-link $node_(s$i) $n1 100000Mb [expr   $i + 1.0]ms DropTail
  $ns duplex-link $node_(k$i) $n2 100000Mb [expr 3*$i + 1.0]ms DropTail

}


#----- Create HSTCP on the forward path -----#
#----- flow class = 0 - 24
# hstcp           = with (=1) or without (=0) hstcp
# hstcp$i         = hstcp tcp connections
# windowOption_ 8 = select HSTCP congestion control
for {set i 0} {$i < $hstcpflows} {incr i} {
  # select if it will high speed  high-speed tcp connection ????
  if {$hstcp==1} {
#    set createType create-highspeed-connection
    set createType create-connection
  } else {
    set createType create-connection
  }
  set hstcp$i [$ns $createType TCP/Sack1 $n1 TCPSink/Sack1 $n2 $i]

  # set tcp congestion control option to be HSTCP
  if {$hstcp==1} {
    [set hstcp$i] set windowOption_ 8
  }

  # set HSTCP parameters
  if {$hstcp==1} {
    [set hstcp$i] set low_window_ $low_window
    [set hstcp$i] set high_window_ $high_window
    [set hstcp$i] set high_p_ $high_p
    [set hstcp$i] set high_decrease_ $high_decrease
  }

  # create ftp traffic in this tcp connection
  set hsftp$i [[set hstcp$i] attach-app FTP]

  # set a random start time for each tcp traffic
  set sec [expr int([$rng uniform 0.1 [expr $endtime/20]])]
  set frac [expr int([$rng uniform 0 25])]
  set starttime $sec.$frac
  $ns at $starttime "[set hsftp$i] start"
  $ns at $endtime "[set hsftp$i] stop"

  # initialize print statistics (util.tcl)
  print-tcpstats [set hstcp$i] $starttime $endtime hstcp$i

  # print this connection start time
  puts "hstcp$i, type: $hstcp  starttime: $starttime"

}


#----- Initiate a periodic report to see simulation evolution -----#
# parm 1 - flow tcp identification
# parm 2 - interval between two reports
# parm 3 - label to put in this report
if {$hstcpflows > 0} {
  timeackReport $hstcp0 1 hstcp0
}


#----- Create regular TCP on forward path with random start time -----#
#----- flow class = 50 - 74
for {set i 0} {$i < $regtcpflows} {incr i} {
  set regtcp$i [$ns create-connection TCP/Sack1 $n1 TCPSink/Sack1 $n2 [expr 50 + $i]]
  set regftp$i [[set regtcp$i] attach-app FTP]
  set sec [expr int([$rng uniform 0.1 [expr $endtime/20]])]
  set frac [expr int([$rng uniform 0 25])]
  set starttime $sec.$frac

  $ns at $starttime "[set regftp$i] start"
  $ns at $endtime "[set regftp$i] stop"

  # complete simulation #
  print-tcpstats [set regtcp$i] $starttime $endtime regtcp$i

  puts "regtcp$i, starttime: $starttime"
}

#----- Select the presence of noise traffic -----#
if {$noisetraf == 1 } {


  #----- Create forward WEB traffic on the bottleneck link (web.tcl) -----#
  #----- flow class = 100
  # param 1 = bootleneck link delay
  # param 2 = total number of sessions in the web pool
  # param 3 = average inter-arrival pages
  # param 4 = page size
  # param 5 = average object size
  # param 6 = flow identification
  # param 7 = forward = 0; backward = 1
  set count 1
  add_web_traffic $delay $web_sessions 4 10 10 100 0


  #----- Create backward WEB traffic on the bottleneck link (web.tcl) -----#
  #----- flow class = 120
  # param 1 = bootleneck link delay
  # param 2 = total number of sessions in the web pool
  # param 3 = average inter-arrival pages
  # param 4 = page size
  # param 5 = average object size
  # param 6 = flow identification
  # param 7 = forward = 0; backward = 1
  set count 1
  add_web_traffic $delay $web_sessions 4 10 10 120 1


  #----- Create Small TCP flows on forward path -----#
  # window_ = 8
  #----- flow class = 140
  for {set i 0} {$i < $smalltcpflows} {incr i} {
    set srcnode [expr int([$rng uniform 0 [expr $max_nrnodes - 0.01] ])]
    set dstnode [expr int([$rng uniform 0 [expr $max_nrnodes - 0.01] ])]
    set tcp [$ns create-connection TCP/Sack1 $node_(s$srcnode) TCPSink/Sack1
             $node_(k$dstnode) 140]
    $tcp set window_ 8
    set ftp [$tcp attach-app FTP]
    set starttime [$rng uniform 0 [expr $endtime/3]]
    set stoptime [$rng uniform [expr ($endtime*2)/3] $endtime]
    $ns at $starttime "$ftp start"
    $ns at $stoptime "$ftp stop"
    # puts "small tcp in fwd direction from $starttime to $stoptime"
  }


  #----- Create Small TCP flows on reverse path -----#
  # window_ = 8
  #----- flow class = 160
  for {set i 0} {$i < $smalltcpflows} {incr i} {
    set srcnode [expr int([$rng uniform 0 [expr $max_nrnodes - 0.01] ])]
    set dstnode [expr int([$rng uniform 0 [expr $max_nrnodes - 0.01] ])]
    set tcp [$ns create-connection TCP/Sack1 $node_(k$srcnode) TCPSink/Sack1
             $node_(s$dstnode) 160]
    $tcp set window_ 8
    set ftp [$tcp attach-app FTP]
    set starttime [$rng uniform 0 [expr $endtime/3]]
    set stoptime [$rng uniform [expr ($endtime*2)/3] $endtime]
    $ns at $starttime "$ftp start"
    $ns at $stoptime "$ftp stop"
    # puts "small tcp in rev direction from $starttime to $stoptime"
  }

}
#-- end of noise traffic --#


#----- Create Small Perturbations that transmit only in their slow-start -----#
#----- They are attached at the bottleneck nodes
#

# perturbation time window in seconds
set pertwin 1.6

```

```
## using a random distribution for initiate each perturbation
  for {set i 0} {$i < $nrpertflows} {incr i} {
    set psrc_($i) [$ns  create-connection TCP/Sack1 $n1 TCPSink/Sack1 $n2 [expr 300 + $i] ]
    $psrc_($i) set max_ssthresh_ 0
    set pertftp_($i) [$psrc_($i) attach-app FTP]

    set sec [expr int([$rng uniform 0.0 [expr $endtime]])]
    set frac [expr int([$rng uniform 0 25])]
    set starttime $sec.$frac

    $ns at $starttime "$pertftp_($i) start"
    $ns at [expr $starttime + $pertwin] "$pertftp_($i) stop"
    $ns at [expr $starttime + $pertwin] "$psrc_($i) reset"
    $ns at $starttime "puts perturbation"
    #print-tcpstats "$psrc_($i)" $starttime [expr $starttime + $pertwin] perttcp$i
  }


#----- Create Perturbations with UDP flows -----#
# Create small perturbations as UDP flows using a Pareto
# distribution to have a long range dependence
# see manual item 33.3 TrafficGenerator
#
# number of pareto flows

for {set i 0} {$i < $nrudpflows} {incr i} {

  set psrc_($i) [new Agent/UDP]
  $psrc_($i) set fid_ 220

  $ns attach-agent $n1 $psrc_($i)

  set prto_($i) [new Application/Traffic/Pareto]
  $prto_($i) set packetSize_ 1460
  $prto_($i) set burst_time_ [expr $onofftime]ms
  $prto_($i) set idle_time_  [expr $onofftime]ms
  $prto_($i) set rate_       30000k
  $prto_($i) set shape_      1.5

  $prto_($i) attach-agent $psrc_($i)

  set psink_($i) [new Agent/Null]
  $ns attach-agent $n2 $psink_($i)

  $ns connect $psrc_($i) $psink_($i)

  set sec [expr int([$rng uniform 0 [expr $endtime/40]])]
  set frac [expr int([$rng uniform 0 25])]
  set starttime $sec.$frac

  $ns at $starttime "$prto_($i) start"

}


##----- Intermediate Link Utilization -----#
# intermediate time interval
set aggint 10

if {$hstcpflows != 0} {
  set TOTKBytesPrev_0    0
  set TOTpdropsPrev_0    0
  set TOTpmarksPrev_0    0
  set TOTparrivalsPrev_0 0
  set fintagg_0 [open aggstats-0.txt w]
  $ns at 0 "print-intaggrlinkutilz 0 $hstcpflows $fmon $aggint"
}

if {$regtcpflows != 0} {
  set TOTKBytesPrev_50    0
  set TOTpdropsPrev_50    0
  set TOTpmarksPrev_50    0
  set TOTparrivalsPrev_50 0
  set fintagg_50 [open aggstats-50.txt w]
  $ns at 0 "print-intaggrlinkutilz 50 $regtcpflows $fmon $aggint"
}

if {$nrudpflows != 0} {
  set TOTKBytesPrev_220    0
  set TOTpdropsPrev_220    0
  set TOTpmarksPrev_220    0
  set TOTparrivalsPrev_220 0
  set fintagg_220 [open aggstats-220.txt w]
  $ns at 0 "print-intaggrlinkutilz 220 $nrudpflows $fmon $aggint"
}


if {$nrpertflows != 0} {
  set TOTKBytesPrev_300    0
  set TOTpdropsPrev_300    0
  set TOTpmarksPrev_300    0
  set TOTparrivalsPrev_300 0
  set fintagg_300 [open aggstats-300.txt w]
  $ns at 0 "print-intaggrlinkutilz 300 $nrpertflows $fmon $aggint"
}

# total traffic
  set TOTKBytesPrev_999    0
  set TOTpdropsPrev_999    0
  set TOTpmarksPrev_999    0
  set TOTparrivalsPrev_999 0
  set fintagg_999 [open aggstats-999.txt w]
  $ns at 0 "print-intaggrlinkutilz 999 0 $fmon $aggint"


##----- Print statistics -----#
set frep [open report.txt w]

#----- HALF simulation -----#
set halftime [expr $endtime / 2]


#----- capture individual flow statistics -----#
 set ffs [open fldata.txt w]
 $ns at $halftime "print-flowstat $fmon"
 $ns at $endtime  "print-flowstat $fmon"


  # print legend
  $ns at $halftime "printlegend"
  # print aggregated flow statistics
  $ns at $halftime "print-aggr $fmon $halftime"

  # print packet drops and link utilization of hstcp
  for {set i 0} {$i < $hstcpflows} {incr i} {
    $ns at $halftime "print-drops $i $fmon"
    $ns at $halftime "print-futilz $i $fmon $halftime"
  }

  # print packet drops and link utilization of tcpflows
```

```
for {set i 50} {$i < [expr 50 + $regtcpflows]} {incr i} {
  $ns at $halftime "print-drops $i $fmon"
  $ns at $halftime "print-futilz $i $fmon $halftime"
}

# print packet drops and link utilization of web traffic on forward path
$ns at $halftime "print-drops 100 $fmon"
$ns at $halftime "print-futilz 100 $fmon $halftime"

# print packet drops and link utilization of web traffic on backward path
$ns at $halftime "print-drops 120 $fmon"
$ns at $halftime "print-futilz 120 $fmon $halftime"

# print packet drops and link utilization of small tcp on forward path
$ns at $halftime "print-drops 140 $fmon"
$ns at $halftime "print-futilz 140 $fmon $halftime"

# print packet drops and link utilization of small tcp on backward path
$ns at $halftime "print-drops 160 $fmon"
$ns at $halftime "print-futilz 160 $fmon $halftime"

# print packet drops and link utilization of perturbation
$ns at $halftime "print-drops 220 $fmon"
$ns at $halftime "print-futilz 220 $fmon $halftime"

# print packet drops and link utilization of tcpflows
for {set i 300} {$i < [expr 300 + $nrpertflows]} {incr i} {
  $ns at $halftime "print-drops $i $fmon"
  $ns at $halftime "print-futilz $i $fmon $halftime"
}


#----- Set the end of simulation -----#
$ns at $halftime "print-gloss"

# print aggregated link utilization and packet lossrate for hstcp
$ns at $halftime "print-aggrlinkutilz 0 $hstcpflows $fmon $halftime"

# print aggregated link utilization and packet lossrate for regtcp
$ns at $halftime "print-aggrlinkutilz 50 $regtcpflows $fmon $halftime"

# print aggregated link utilization and packet lossrate for perturbation
$ns at $halftime "print-aggrlinkutilz 220 $nrudpflows $fmon $halftime"

# print aggregated link utilization and packet lossrate for perturbation
$ns at $halftime "print-aggrlinkutilz 300 $nrpertflows $fmon $halftime"

# get aggregated hstcp drops and marks packets
$ns at $halftime "print-aggrdpmk 0 $hstcpflows $fmon $halftime"

# get aggregated regtcp drops and marks packets
$ns at $halftime "print-aggrdpmk 50 $regtcpflows $fmon $halftime"



#----- END simulation -----#
# print legend
#$ns at $endtime "printlegend"
# print aggregated flow statistics
$ns at $endtime "print-aggr $fmon $endtime"

# print packet drops and link utilization of hstcp
for {set i 0} {$i < $hstcpflows} {incr i} {
  $ns at $endtime "print-drops $i $fmon"
  $ns at $endtime "print-futilz $i $fmon $endtime"
  }

  # print packet drops and link utilization of tcpflows
  for {set i 50} {$i < [expr 50 + $regtcpflows]} {incr i} {
    $ns at $endtime "print-drops $i $fmon"
    $ns at $endtime "print-futilz $i $fmon $endtime"
  }

  # print packet drops and link utilization of web traffic on forward path
  $ns at $endtime "print-drops 100 $fmon"
  $ns at $endtime "print-futilz 100 $fmon $endtime"

  # print packet drops and link utilization of web traffic on backward path
  $ns at $endtime "print-drops 120 $fmon"
  $ns at $endtime "print-futilz 120 $fmon $endtime"

  # print packet drops and link utilization of small tcp on forward path
  $ns at $endtime "print-drops 140 $fmon"
  $ns at $endtime "print-futilz 140 $fmon $endtime"

  # print packet drops and link utilization of small tcp on backward path
  $ns at $endtime "print-drops 160 $fmon"
  $ns at $endtime "print-futilz 160 $fmon $endtime"

  # print packet drops and link utilization of perturbation
  $ns at $endtime "print-drops 220 $fmon"
  $ns at $endtime "print-futilz 220 $fmon $endtime"

  # print packet drops and link utilization of tcpflows
  for {set i 300} {$i < [expr 300 + $regtcpflows]} {incr i} {
    $ns at $endtime "print-drops $i $fmon"
    $ns at $endtime "print-futilz $i $fmon $endtime"
  }


  #----- Set the end of simulation -----#
  $ns at [expr $endtime + 1.0] "print-gloss"
  $ns at [expr $endtime + 1.0] "gnlreport gnlrpt.txt"
  $ns at [expr $endtime + 2.0] "finish"

  # print aggregated link utilization and packet lossrate for hstcp
  $ns at $endtime "print-aggrlinkutilz 0 $hstcpflows $fmon $endtime"

  # print aggregated link utilization and packet lossrate for regtcp
  $ns at $endtime "print-aggrlinkutilz 50 $regtcpflows $fmon $endtime"

  # print aggregated link utilization and packet lossrate for perturbation
  $ns at $endtime "print-aggrlinkutilz 220 $nrudpflows $fmon $endtime"

  # print aggregated link utilization and packet lossrate for perturbation
  $ns at $endtime "print-aggrlinkutilz 300 $nrpertflows $fmon $endtime"

  # get aggregated hstcp drops and marks packets
  $ns at $endtime "print-aggrdpmk 0 $hstcpflows $fmon $endtime"

  # get aggregated regtcp drops and marks packets
  $ns at $endtime "print-aggrdpmk 50 $regtcpflows $fmon $endtime"

#----- Create trace files (manual 22, ~/tcl/lib/ns-lib.tcl)-----#
# set tf [open qtrace.out w]
# $ns at 0 "$ns trace-queue $n1 $n2 $tf"
# $ns at 0 "$ns trace-queue $n2 $n1 $tf"


#----- Start the simulation -----#
```

```
puts "Starting Simulation..."
$ns run

```

A.2 web.tcl

```
###################################
# WEB Traffic Generator - web.tcl #
# Evandro de Souza - Apr/2002     #
# LBNL/Unicamp/Embrapa            #
###################################


# Add web nodes (architecture and links)
# bdel      = general delay
# randomize = choose if link day for web nodes will be fixed or random
# dir       = 0 - server with N1; 1 - server with N2
# n1        - node 1 of the bottleneck link
# n2        - node 2 of the bottleneck link
# sf_ sb_   - server web nodes declared as global variable (forward / backward)
# rf_ rb_   - client web nodes declared as global variable (forward / backward)
# count     - "static" variable to control how many web nodes were included
proc add_web_nodes {bdel randomize dir} {
  global ns n1 n2
  global sf_ rf_
  global sb_ rb_
  global count

  # choose web link delay
  if {$randomize == 0} {
    set x [expr $bdel/2]ms
    set y [expr $bdel/2]ms
  } else {
    set x [ns-random]
    set y [ns-random]
    set x [expr $bdel*($x/2147483647.0)]ms
    set y [expr $bdel*($y/2147483647.0)]ms
  }

  # set web nodes
  set i $count

  # forward web traffic
  if {$dir == 0} {
    set sf_($i) [$ns node]
    set rf_($i) [$ns node]
    $ns duplex-link $sf_($i) $n1 10000Mb $x DropTail
    $ns duplex-link $rf_($i) $n2 10000Mb $y DropTail
  # backward web traffic
  } else {
    set sb_($i) [$ns node]
    set rb_($i) [$ns node]
    $ns duplex-link $sb_($i) $n2 10000Mb $x DropTail
    $ns duplex-link $rb_($i) $n1 10000Mb $y DropTail
  }
  incr count
}


# Add web traffic and set it behavior (manual item 34.4.5)
# bdel   = general delay
# nums   = total number of sessions in the web pool
# ip     = average inter-arrival pages
# ps     = page size
# os     = average object size
# flowid = flow identification for all web flows in the monitor
# dir    = servers position (0 = linked in n1) and (1 = linked in n2)
# n1        - node 1 of the bottleneck link
# n2        - node 2 of the bottleneck link
# count     - "static" variable to control how many web nodes were included
# pool      - handler to PagePool class
# numWeb    - number of clients/servers web and web sessions
# i         - session index
# numPage   - total number of pages per session
# interPage - random variable that generates page inter-arrival time
# pageSize  - random variable that generates number of objects per page
# interObj  - random variable that generates object inter-arrival time
# objSize   - random variable that generates object size
proc add_web_traffic {bdel nums ip ps os flowid dir} {
  global ns n1 n2
  global sf_ rf_
  global sb_ rb_
  global count
  global pool

  # create a pool of web client/server with the same flowid
  set numWeb 10
  PagePool/WebTraf set FID_ASSIGNING_MODE_ 2
  # (manual item 34.4.5)
  set pool [new PagePool/WebTraf]
  # set total number of clients
  $pool set-num-client $numWeb
  # set total number of servers
  $pool set-num-server $numWeb
  $pool set sameFid_ $flowid

  # set all client and servers web nodes and associate which node is server and which
  # is client
  for {set i 0} {$i < $numWeb} {incr i} {
    add_web_nodes $bdel 1 $dir
    # forward
    if {$dir == 0} {
      $pool set-server $i $sf_([expr $count - 1])
      $pool set-client $i $rf_([expr $count - 1])
    # backward
    } else {
      $pool set-server $i $sb_([expr $count - 1])
      $pool set-client $i $rb_([expr $count - 1])
    }

  }

  # set the total number of of sessions in the WebTraf pool
  $pool set-num-session $nums
  # set the total number of pages per session
  set numPage 1000
  # set each web session with random variables
  for {set i 0} {$i < $nums} {incr i} {
    # set interPage with a exponetial distribution and average ip (manual item 21.2)
    set interPage [new RandomVariable/Exponential]
    $interPage set avg_ $ip
    # set pageSize with a constant distribution value ps (manual item 21.2)
    set pageSize [new RandomVariable/Constant]
    $pageSize set val_ $ps
    # set interObj with exponetial distribution and average 0.01 (manual item 21.2)
    set interObj [new RandomVariable/Exponential]
    $interObj set avg_ [expr 0.01]
    # set objSize with paretto distribution and average os and shape 1.2 (manual item 21.2)
    set objSize [new RandomVariable/ParetoII]
    $objSize set avg_ $os
    $objSize set shape_ 1.2

    # Add RedHat 8.0
    $pool set req_trace_ 0
    $pool set resp_trace_ 0

    # create a pool of web sessions with lanch time = 0
    $pool create-session $i $numPage 0 $interPage $pageSize $interObj $objSize
  }
}

```

A.3 utils.tcl

```
####################################
# Statistics Utilities - utils.tcl #
# Evandro de Souza - Apr/2002      #
# LBNL/Unicamp/Embrapa             #
####################################


#----- Print general loss in the bottleneck link -----#
# pdrops     = cumulative dropped packets
# parrivals  = cumulative arrived packets
# lossrate   = pdrops/parrivals
# bdpartures = cumulative departured bytes
# qmon  - queue monitor handler
# frep  - file report pointer
# qflag - queue monitor flag
proc print-gloss {} {
  global qmon ns frep qflag

  $ns flush-trace
  if {$qflag == 1} {
    set now [$ns now]
    puts "at $now, bdeparture=[$qmon set bdepartures_]"
    puts "at $now, lossrate=[expr ("[$qmon set pdrops_].0")/[$qmon set parrivals_]]"

    puts $frep "at $now, bdeparture=[$qmon set bdepartures_]"
    puts $frep "at $now, lossrate=[expr ("[$qmon set pdrops_].0")/[$qmon set parrivals_]]"
  }

}

#----- Flush all trace data when finish and print general stitistics -----#
# tf - trace file pointer
proc finish {} {
  global ns tf namtf

#  close $tf
#  close $namtf
  exit 0
}


```

```
#----- Save queue statistics (manual item 7.3) (~/tools/queue-monitor.h)-----#
# now         = current simulation time
# size        = instatenous queue size in bytes
# pkts        = instatenous queue size in packets
# parrivals   = running queue size in packets
# barrivals   = running queue size in bytes
# pdepartures = running total of packets that have departed (not dropped)
# bdepartures = running total of bytes contained in packets that have departed
#               (not dropped)
# pdrops      = total number of packets dropped
# bdrops      = total number of bytes dropped
# pmarks      = total ecn marked packets
# fp   - file pointer
# qmon - queue monitor handler
proc printqueue {fp} {
  global qmon ns

  set now [$ns now]
  puts $fp "[format %.2f [$ns now]] [$qmon set size_] [$qmon set pkts_]
[$qmon set parrivals_] [$qmon set barrivals_] [$qmon set pdepartures_]
[$qmon set bdepartures_] [$qmon set pdrops_] [$qmon set bdrops_] [$qmon set pmarks_]"
}


#----- Collect queue statistics and schedule new collect time -----#
# starttime  - start collecting time
# finishtime - simulation final time
# qfp - queue file pointer
proc print-qstats {starttime finishtime fname} {
  global ns

  set qfp [open $fname.out w]
  for {set i 0} {[expr $starttime + ($i*0.1)] < $finishtime} {incr i} {
    set printtime [expr $i*0.1]
    $ns at [expr $starttime + $printtime] "printqueue $qfp"
  }
  $ns at $finishtime "printqueue $qfp"
}


#----- Collect TCP statistics (manual item 10.8) (~/tcp/tcp.h) -----#
# cwnd_          = current window size
# ndatapack_     = number of data packets sent
# ndatabytes_    = number of data bytes sent
# nackpack_      = number of acknowledge packets received
# nrexmit_       = number of retransmit timeout
# nrexmitpack_   = number of retransmited packets
# nrexmitbytes_  = number of retransmited bytes
# necnresponses_ = number of ecn responses
# ncwndcuts_     = number of times cwnd cuts
# maxseq_        = maximum sequence number sent
# ack_           = highest ack seen by the receiver
# dupacks_       = number of duplicate acks
# ssthresh_      = slow start threshold
# rtt            = round-trip time estimate
# srtt           = smoothed round-trip time estimate
# rttvar         = round-trip time mean deviation estimate
# fp  - file pointer
# tcp - TCP connection identification
proc printtcp {fp tcp} {
  global ns

  set now [format "%.1f" [$ns now]]
  set now [$ns now]
  puts $fp "[format %.2f [$ns now]] [$tcp set cwnd_] [$tcp set ndatapack_]
```

```
[$tcp set ndatabytes_] [$tcp set nackpack_] [$tcp set nrexmit_] [$tcp set nrexmitpack_]
[$tcp set nrexmitbytes_] [$tcp set necnresponses_] [$tcp set ncwndcuts_]
[$tcp set maxseq_] [$tcp set ack_] [$tcp set dupacks_] [$tcp set ssthresh_]
[$tcp set rtt_] [$tcp set srtt_] [$tcp set rttvar_]"
}


#----- Collect TCP statistics and schedule a new collect time -----#
# flow       - TCP flow identification
# starttime  - time of the last print
# finishtime - simulation final time
proc print-tcpstats {flow starttime finishtime label} {
  global ns

  set fp [open "tcp-$label.out" w]
  for {set i 0} {[expr $starttime + ($i*0.05)] < $finishtime} {incr i} {
    set printtime [expr $i*0.05]
    $ns at [expr $starttime + $printtime] "printtcp $fp $flow"
  }
  $ns at $finishtime "printtcp $fp $flow"
}


#----- Periodic report of a flow, to follow time and sequence number -----#
# ack_     = highest ack seen by the receiver
# tcpSrc   - flow identification
# interval - time interval to report
# label    - label to print in the report
proc timeackReport { tcpSrc interval label } {
  global ns

  $ns at [expr [$ns now]+$interval] "timeackReport $tcpSrc $interval $label"
  puts $label/time=[$ns now]/ack=[$tcpSrc set ack_]
}


#----- Aggregate Flow Report -----#
# pdrops_      = total number of packets dropped (manual item 7.3)
# pmarks       = total ecn marked packets
# bdepartures_ = running total of bytes contained in packets that have departed
#                (not dropped)
# pdepartures_ = running total of bytes packets that have departed (not dropped)
# fmon         - flow monitor
# time         - time for this report
# bandwidth    -> bottleneck link bandwidth
# frep         -> report file pointer
# halftime     -> half simulation time
# drops_1      -> drops in the simulation 1st half
# drops_t      -> drops in the whole simulation
# marks_1      -> marks in the simulation 1st half
# marks_t      -> marks in the whole simulation
# packets_1    -> packets sent in the simulation 1st half
# packets_t    -> packets sent in the whole simulation
# tlu_1        -> total link utilization in the simulation 1st half
# tlu_t        -> total link utilization in the whole simulation
proc print-aggr { fmon time } {
  global bandwidth frep halftime drops_1 drops_t marks_1 marks_t packets_1 packets_t
tlu_1 tlu_t

  set bytes [$fmon set bdepartures_]
  set bytesDbl [ns-int64todbl $bytes]
  set Kbytes [expr $bytesDbl / 1000 ]
  set bandwidthToKBytes  [expr 1000 * $time / 8 ]
  set possibleKBytes [expr $bandwidth * $bandwidthToKBytes ]

  puts " "
  puts "time: [format %.2f $time] aggregate per-link drops  [$fmon set pdrops_]"
  puts "time: [format %.2f $time] aggregate per-link marks  [$fmon set pmarks_]"
  puts "time: [format %.2f $time] aggregate per-link pkts   [$fmon set pdepartures_]"
  puts "time: [format %.2f $time] aggregate per-link Kbytes [format %.2f $Kbytes]"
  puts "time: [format %.2f $time] aggregate per-link utilz  [format %.2f
                                       [expr $Kbytes * 100 / $possibleKBytes ]](%)"

  puts $frep " "
  puts $frep "time: [format %.2f $time] aggregate per-link drops  [$fmon set pdrops_]"
  puts $frep "time: [format %.2f $time] aggregate per-link marks  [$fmon set pmarks_]"
  puts $frep "time: [format %.2f $time] aggregate per-link pkts   [$fmon set pdepartures_]"
  puts $frep "time: [format %.2f $time] aggregate per-link Kbytes [format %.2f $Kbytes]"
  puts $frep "time: [format %.2f $time] aggregate per-link utilz  [format %.2f
                                             [expr $Kbytes * 100 / $possibleKBytes ]](%)"

  if {$time == $halftime} {
    set drops_1   [$fmon set pdrops_]
    set marks_1   [$fmon set pmarks_]
    set packets_1 [$fmon set pdepartures_]
    set tlu_1     [format %.2f [expr $Kbytes * 100 / $possibleKBytes ]]

  } else {
    set drops_t   [$fmon set pdrops_]
    set marks_t   [$fmon set pmarks_]
    set packets_t [$fmon set pdepartures_]
    set tlu_t     [format %.2f [expr $Kbytes * 100 / $possibleKBytes ]]
  }

}


#----- Packet Drop Report per flow (manual item 7.3) -----#
# bdepartures_ = running total of bytes contained in packets that have departed
#                (not dropped)
# pdepartures_ = running total of bytes packets that have departed (not dropped)
# pdrops_      = total number of packets dropped
# pmarks       = total ecn marked packets
# fid  - flow identification
# fmon - flow monitor
# frep  - file report pointer
proc print-drops { fid fmon } {
  global frep

  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]

    puts "class: $fid per-link pkts [$flow set pdepartures_] Kbytes [format %.2f
[expr $bytesDbl / 1000]] drops [$flow set pdrops_] marks [$flow set pmarks_]"

    puts $frep "class: $fid per-link pkts [$flow set pdepartures_] Kbytes [format %.2f
[expr $bytesDbl / 1000]] drops [$flow set pdrops_] marks [$flow set pmarks_]"

  }
}


#----- Link Utilization per flow (manual item 7.3) -----#
# bdepartures_ = running total of bytes contained in packets that have departed
#                (not dropped)
# pdrops_      = total number of packets dropped
# parrivals    = Running total of packets that have arrived
# endtime      = simulation time
# bandwidth    = bottleneck link bandwidth
# fid  - flow identification
# fmon - flow monitor
# frep  - file report pointer
proc print-futilz {fid fmon time} {
  global bandwidth frep

  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]
    set Kbytes [expr $bytesDbl / 1000 ]
    set bandwidthToKBytes  [expr 1000 * $time / 8 ]
    set possibleKBytes [expr $bandwidth * $bandwidthToKBytes ]
    set lossrate [expr ("[$flow set pdrops_].0")/[$flow set parrivals_]]

    puts "class: $fid per-link utilization [format %.2f [expr $Kbytes * 100 /
$possibleKBytes]](%)  lossrate = $lossrate"

    puts $frep "class: $fid per-link utilization [format %.2f [expr $Kbytes * 100 /
$possibleKBytes]](%)  lossrate = $lossrate"

  }
}


#----- Agregated Link Utilization for HSTCP and REGTCP (manual item 7.3) -----#
# bdepartures_ = running total of bytes contained in packets that have departed
#                (not dropped)
# pdrops_      = total number of packets dropped
# parrivals    = Running total of packets that have arrived
# endtime      = simulation time
# bandwidth    = bottleneck link bandwidth
# ffid     - first flow identification
# nrflows  - number of flows in this class
# fmon     - flow monitor
# agghs_1   -> aggregate hstcp flows link utilization in the 1st simulation half
# agghs_t   -> aggregate hstcp flows link utilization in whole simulation
# aggreg_1  -> aggregate regtcp flows link utilization in the 1st simulation half
# aggreg_t  -> aggregate regtcp flows link utilization in whole simulation
# avghs_1   -> average hstcp flows link utilization in the 1st simulation half
# avghs_t   -> average hstcp flows link utilization in whole simulation
# avgreg_1  -> average regtcp flows link utilization in the 1st simulation half
# avgreg_t  -> average regtcp flows link utilization in whole simulation

proc print-aggrlinkutilz {ffid nrflows fmon time} {
  global bandwidth frep halftime agghs_1 agghs_t aggreg_1 aggreg_t aggpert_1 aggpert_t
avghs_1 avghs_t avgreg_1 avgreg_t

  set linkutilz    0
  set linkutilzavg 0
  set TOTpdrops    0
  set TOTparrivals 0

  set bandwidthToKBytes  [expr 1000 * $time / 8 ]
  set possibleKBytes [expr $bandwidth * $bandwidthToKBytes ]

  for {set i $ffid} {$i < [expr $ffid + $nrflows]} {incr i} {
    set fcl [$fmon classifier]; # flow classifier
    set flow [$fcl lookup auto 0 0 $i]
    if {$flow != "" } {
      set bytes [$flow set bdepartures_]
      set bytesDbl [ns-int64todbl $bytes]
      set Kbytes [expr $bytesDbl / 1000 ]
      set aux [expr $linkutilz + [expr $Kbytes * 100 / $possibleKBytes] ]
      set linkutilz $aux

      set aux [ns-int64todbl [expr $TOTpdrops + [$flow set pdrops_] ] ]
      set TOTpdrops [ns-int64todbl $aux]

      set aux [ns-int64todbl [expr $TOTparrivals + [$flow set parrivals_] ] ]
      set TOTparrivals [ns-int64todbl $aux]
    }

  }

  if { $nrflows != 0 } {
    set linkutilzavg [expr $linkutilz / $nrflows]
  }

  if {$TOTparrivals != 0} {
    set lossrate [expr $TOTpdrops/$TOTparrivals]
  } else {
      set lossrate 0.0
  }

  puts "Aggr Class $ffid :: link utilz = [format %.2f $linkutilz] (%)  lossrate = $lossrate"

  puts $frep "Aggr Class $ffid :: link utilz = [format %.2f $linkutilz] (%)  lossrate =
$lossrate"


  #----- aggregate -----#
  if {$ffid == 0 && $time == $halftime } {
        set agghs_1 [format %.2f $linkutilz]
  }
  if {$ffid == 50 && $time == $halftime } {
        set aggreg_1 [format %.2f $linkutilz]
  }
  if {$ffid == 300 && $time == $halftime } {
        set aggpert_1 [format %.2f $linkutilz]
  }
  if {$ffid == 0 && $time > $halftime } {
        set agghs_t [format %.2f $linkutilz]
  }
  if {$ffid == 50 && $time > $halftime } {
        set aggreg_t [format %.2f $linkutilz]
  }
  if {$ffid == 300 && $time > $halftime } {
        set aggpert_t [format %.2f $linkutilz]
  }

  #----- average -----#
  if {$ffid == 0 && $time == $halftime } {
        set avghs_1 [format %.2f $linkutilzavg]
  }
  if {$ffid == 50 && $time == $halftime } {
        set avgreg_1 [format %.2f $linkutilzavg]
  }
  if {$ffid == 0 && $time > $halftime } {
        set avghs_t [format %.2f $linkutilzavg]
  }
  if {$ffid == 50 && $time > $halftime } {
        set avgreg_t [format %.2f $linkutilzavg]
  }

}

```

```


#----- General Simulation Report -----#
# fname = file name to save this report
# fp    = file pointer
# tlu_2        - Total link utilization in 2nd half simulation time
# agghs_2      - Aggregate hstcp link utilization in the 2nd half of simulation time
# aggreg_2     - Aggregate regtcp link utilization in the 2nd half of simulation time
# avghs_2      - Average hstcp link utilization in the 2nd half of simulation time
# avgreg_2     - Average regtcp link utilization in the 2nd half of simulation time
# drops_2      - Drops in the 2nd half of simulation time
# agghsregrt_2 - Ratio between aggregate hstcp and aggregate regtcp in the 2nd half of
#                simulation time
# marks_2      - Marks in the 2nd half of simulation time
# dpmk_1       - Sum of drops + marks in the 1st half of simulation time
# dpmk_t       - Sum of drops + marks in the whole simulation time
# dpmk_2       - Sum of drops + marks in the 2nd half of simulation time
# packets_2    - Packets sent in the 2nd half of simulation time
# lr_2         - Loss Rate in the 2nd half of simulation time
proc gnlreport { fname } {
  global hstcpflows regtcpflows tlu_1 tlu_t agghs_1 agghs_t aggreg_1 aggreg_t aggpert_1
aggpert_t avghs_1 avghs_t avgreg_1 avgreg_t drops_1 marks_1 drops_t marks_t packets_1
packets_t agghsdp_1 agghsmk_1 agghsdp_t agghsmk_t aggregdp_1 aggregdp_t aggregmk_1
aggregmk_t

  # variables declaration
  set tlu_2        0
  set agghs_2      0
  set aggreg_2     0
  set aggpert_2    0
  set avghs_2      0
  set avgreg_2     0
  set drops_2      0
  set marks_2      0
  set dpmk_1       0
  set dpmk_t       0
  set dpmk_2       0
  set packets_2    0
  set lr_2         0
  set agghsregrt_2 0
  set avghsregrt_2 0
  set agghsdp_2    0
  set agghsmk_2    0
  set aggregdp_2   0
  set aggregmk_2   0

  # variables calculus
  set tlu_2        [format %.2f [expr 2 * $tlu_t    - $tlu_1]]
  set agghs_2      [format %.2f [expr 2 * $agghs_t  - $agghs_1]]
  set aggreg_2     [format %.2f [expr 2 * $aggreg_t - $aggreg_1]]
  set aggpert_2    [format %.2f [expr 2 * $aggpert_t - $aggpert_1]]
  set avghs_2      [format %.2f [expr 2 * $avghs_t  - $avghs_1]]
  set avgreg_2     [format %.2f [expr 2 * $avgreg_t - $avgreg_1]]

  if { $aggreg_2 != 0 } {
    set agghsregrt_2 [format %.10f [expr $agghs_2 / $aggreg_2]]
  }

  if { $avgreg_2 != 0 } {
    set avghsregrt_2 [format %.10f [expr $avghs_2 / $avgreg_2]]
  }


  set drops_2      [expr $drops_t   - $drops_1]
  set marks_2      [expr $marks_t   - $marks_1]
  set dpmk_1       [expr $drops_1   + $marks_1]
  set dpmk_t       [expr $drops_t   + $marks_t]
  set dpmk_2       [expr $dpmk_t    - $dpmk_1]
  set packets_2    [expr $packets_t - $packets_1]

  set DBLdpmk_2    [ns-int64todbl $dpmk_2]
  set DBLpackets_2 [ns-int64todbl $packets_2]
  set lr_2         [format %.10f [expr $DBLdpmk_2 / $DBLpackets_2]]

  set agghsdp_2    [expr $agghsdp_t - $agghsdp_1]
  set agghsmk_2    [expr $agghsmk_t - $agghsmk_1]
  set aggregdp_2   [expr $aggregdp_t - $aggregdp_1]
  set aggregmk_2   [expr $aggregmk_t - $aggregmk_1]


  # save values in the file
  set fp [open $fname w]
  puts $fp "$hstcpflows $regtcpflows $tlu_1 $tlu_t $tlu_2 $agghs_1 $agghs_t $agghs_2
$aggreg_1 $aggreg_t $aggreg_2 $agghsregrt_2 $avghs_1 $avghs_t $avghs_2 $avgreg_1
$avgreg_t $avgreg_2 $avghsregrt_2 $drops_1 $drops_t $drops_2 $marks_1 $marks_t
$marks_2 $dpmk_1 $dpmk_t $dpmk_2 $packets_1 $packets_t $packets_2 $lr_2 $agghsdp_1
$agghsdp_t $agghsdp_2 $agghsmk_1 $agghsmk_t $agghsmk_2 $aggregdp_1 $aggregdp_t
$aggregdp_2 $aggregmk_1 $aggregmk_t $aggregmk_2 $aggpert_1 $aggpert_t $aggpert_2"
  close $fp

}


#----- Aggregate Drops and Marks for HSTCP and REGTCP -----#
# ffid    - first flow identification
# nrflows - number of flows in this class
# fmon    - flow monitor
# time    - phase of simulation to collect data
# halftime = half time of the simulation
# agg*     = global variables (see behavior.tcl)

proc print-aggrdpmk { ffid nrflows fmon time } {
  global halftime agghsdp_1 agghsmk_1 agghsdp_t agghsmk_t aggregdp_1 aggregdp_t
aggregmk_1 aggregmk_t

  #----- if there is flows in this class -----#
  if { $nrflows > 0 }  {

    set TOTpdrops 0
    set TOTpmarks 0

    #----- get each flow in this class -----#
    for {set i $ffid} {$i < [expr $ffid + $nrflows]} {incr i} {
      set fcl [$fmon classifier]; # flow classifier
      set flow [$fcl lookup auto 0 0 $i]
      if {$flow != "" } {

        set aux [ns-int64todbl [expr $TOTpdrops + [$flow set pdrops_] ] ]
        set TOTpdrops [format %.0f [ns-int64todbl $aux]]

        set aux [ns-int64todbl [expr $TOTpmarks + [$flow set pmarks_] ] ]
        set TOTpmarks [format %.0f [ns-int64todbl $aux]]

      }
    }

    #----- put the results in the global variable -----#
    if {$ffid == 0 && $time == $halftime } {
          set agghsdp_1 $TOTpdrops
          set agghsmk_1 $TOTpmarks
    }
    if {$ffid == 50 && $time == $halftime } {
          set aggregdp_1 $TOTpdrops
          set aggregmk_1 $TOTpmarks
    }
    if {$ffid == 0 && $time > $halftime } {
          set agghsdp_t $TOTpdrops
          set agghsmk_t $TOTpmarks
    }
    if {$ffid == 50 && $time > $halftime } {
          set aggregdp_t $TOTpdrops
          set aggregmk_t $TOTpmarks
    }

  }

}

#----- Intermidiate Aggregate link Utilization ----- (Sep/12/2002) #
# bdepartures_     = running total of bytes contained in packets that have departed
# (not dropped)
# pdrops_          = total number of packets dropped
# parrivals        = Running total of packets that have arrived
# bandwidth        = bottleneck link bandwidth
# endtime          = total simulation time
# TOTKBytesPrev    = previous total Kbytes transmitted
# TOTpdropsPrev    = previous total packets dropped
# TOTparrivalsPrev = previous total packets arrivals
# fintagg          = file pointer to intermidiate aggregate link utilization
# ffid     - first flow identification
# nrflows  - number of flows in this class
# fmon     - flow monitor


proc print-intaggrlinkutilz {ffid nrflows fmon inttime} {
  global ns bandwidth endtime qmon TOTKBytesPrev_0 TOTpdropsPrev_0 TOTpmarksPrev_0
TOTparrivalsPrev_0 fintagg_0 TOTKBytesPrev_50 TOTpdropsPrev_50 TOTparrivalsPrev_50
TOTpmarksPrev_50 fintagg_50 TOTKBytesPrev_220 TOTpdropsPrev_220 TOTparrivalsPrev_220
TOTpmarksPrev_220 fintagg_220 TOTKBytesPrev_300 TOTpdropsPrev_300 TOTparrivalsPrev_300
TOTpmarksPrev_300 fintagg_300 TOTKBytesPrev_999 TOTpdropsPrev_999 TOTparrivalsPrev_999
TOTpmarksPrev_999 fintagg_999

  if {$ffid == 0} {
    set TOTKBytesPrev     $TOTKBytesPrev_0
    set TOTpdropsPrev     $TOTpdropsPrev_0
    set TOTpmarksPrev     $TOTpmarksPrev_0
    set TOTparrivalsPrev  $TOTparrivalsPrev_0
    set fintagg           $fintagg_0
  }

  if {$ffid == 50} {
    set TOTKBytesPrev     $TOTKBytesPrev_50
    set TOTpdropsPrev     $TOTpdropsPrev_50
    set TOTpmarksPrev     $TOTpmarksPrev_50
    set TOTparrivalsPrev  $TOTparrivalsPrev_50
    set fintagg           $fintagg_50
  }

  if {$ffid == 220} {
    set TOTKBytesPrev     $TOTKBytesPrev_220
    set TOTpdropsPrev     $TOTpdropsPrev_220
    set TOTpmarksPrev     $TOTpmarksPrev_220
    set TOTparrivalsPrev  $TOTparrivalsPrev_220
    set fintagg           $fintagg_220
  }

  if {$ffid == 300} {
    set TOTKBytesPrev     $TOTKBytesPrev_300
    set TOTpdropsPrev     $TOTpdropsPrev_300
    set TOTpmarksPrev     $TOTpmarksPrev_300
    set TOTparrivalsPrev  $TOTparrivalsPrev_300
    set fintagg           $fintagg_300
  }

  if {$ffid == 999} {
    set TOTKBytesPrev     $TOTKBytesPrev_999
    set TOTpdropsPrev     $TOTpdropsPrev_999
    set TOTpmarksPrev     $TOTpmarksPrev_999
    set TOTparrivalsPrev  $TOTparrivalsPrev_999
    set fintagg           $fintagg_999
  }


  set linkutilz    0
  set TOTparrivals 0
  set TOTpdrops    0
  set TOTpmarks    0
  set TOTparrivals 0
  set TOTKbytes    0

  set bandwidthToKBytes  [expr 1000 * $inttime / 8 ]
  set possibleKBytes [expr $bandwidth * $bandwidthToKBytes ]

  for {set i $ffid} {$i < [expr $ffid + $nrflows]} {incr i} {
    set fcl [$fmon classifier]; # flow classifier
    set flow [$fcl lookup auto 0 0 $i]
    if {$flow != "" } {
      set bytes [$flow set bdepartures_]
      set bytesDbl [ns-int64todbl $bytes]
      set Kbytes [expr $bytesDbl / 1000 ]
      set TOTKbytes [expr $TOTKbytes + $Kbytes]

      set aux [ns-int64todbl [expr $TOTpdrops + [$flow set pdrops_] ] ]
      set TOTpdrops [ns-int64todbl $aux]

      set aux [ns-int64todbl [expr $TOTpmarks + [$flow set pmarks_] ] ]
      set TOTpmarks [ns-int64todbl $aux]

      set aux [ns-int64todbl [expr $TOTparrivals + [$flow set parrivals_] ] ]
      set TOTparrivals [ns-int64todbl $aux]

    }

  }

  if {$ffid == 999} {
    set TOTpdrops    [ns-int64todbl "[$qmon set pdrops_].0" ]
    set TOTpmarks    [ns-int64todbl "[$qmon set pmarks_].0" ]
    set TOTparrivals [ns-int64todbl "[$qmon set parrivals_].0" ]

    set bytes     [$qmon set bdepartures_]
    set bytesDbl  [ns-int64todbl $bytes]
    set Kbytes    [expr $bytesDbl / 1000 ]
    set TOTKbytes [expr $TOTKbytes + $Kbytes]
  }


  set INTKbytes [expr $TOTKbytes - $TOTKBytesPrev]
  set aux [expr $INTKbytes * 100 / $possibleKBytes]
  set linkutilz $aux

  set INTpdrops [expr $TOTpdrops - $TOTpdropsPrev]
  set INTpmarks [expr $TOTpmarks - $TOTpmarksPrev]

  set INTparrivals [expr $TOTparrivals - $TOTparrivalsPrev]


    puts $fintagg "[format %.2f [$ns now]] [format %.2f $linkutilz]
[format %.2f $INTKbytes] [format %0.f $INTpdrops] [format %0.f $INTpmarks]
[format %2.f $INTparrivals]"


  if {$ffid == 0} {
    set TOTKBytesPrev_0     $TOTKbytes
    set TOTpdropsPrev_0     $TOTpdrops
    set TOTpmarksPrev_0     $TOTpmarks
    set TOTparrivalsPrev_0  $TOTparrivals
  }

  if {$ffid == 50} {
    set TOTKBytesPrev_50     $TOTKbytes
    set TOTpdropsPrev_50     $TOTpdrops
    set TOTpmarksPrev_50     $TOTpmarks
    set TOTparrivalsPrev_50  $TOTparrivals
  }

  if {$ffid == 220} {
    set TOTKBytesPrev_220     $TOTKbytes
    set TOTpdropsPrev_220     $TOTpdrops
    set TOTpmarksPrev_220     $TOTpmarks
    set TOTparrivalsPrev_220  $TOTparrivals
  }


  if {$ffid == 300} {
    set TOTKBytesPrev_300     $TOTKbytes
    set TOTpdropsPrev_300     $TOTpdrops
    set TOTpmarksPrev_300     $TOTpmarks
    set TOTparrivalsPrev_300  $TOTparrivals
  }

  if {$ffid == 999} {
    set TOTKBytesPrev_999     $TOTKbytes
    set TOTpdropsPrev_999     $TOTpdrops
    set TOTpmarksPrev_999     $TOTpmarks
    set TOTparrivalsPrev_999  $TOTparrivals
  }


    set nexttime [expr [$ns now] + $inttime]

  if {$nexttime < $endtime} {
    $ns at [expr [$ns now]+$inttime] "print-intaggrlinkutilz $ffid $nrflows $fmon $inttime"
  }

}



#----- Print Flow Statistics -----#
# fmon    - flow monitor
# bandwidth  - bottleneck link bandwidth
# hstcpflows - nr hstcp flows
# regtcflows - nr regtcp flows
# ffs        - file to save flowstat
```

```
proc print-flowstat { fmon } {
  global ns bandwidth hstcpflows regtcpflows nrpertflows ffs qmon

  # calculus of possible bytes to sent
  set bandwidthToKBytes  [expr 1000 * [$ns now] / 8 ]
  set possibleKBytes [expr $bandwidth * $bandwidthToKBytes ]

  # array of results
  set result [format %.2f [$ns now]]
  set aux "$result $hstcpflows $regtcpflows"
  set result $aux

  # HSTCP Flows
  for {set fid 0} {$fid < [expr 0 + $hstcpflows]} {incr fid} {
    # extract data for this particular flow
    set fcl [$fmon classifier]; # flow classifier
    set flow [$fcl lookup auto 0 0 $fid]
    if {$flow != "" } {
      set bytes [$flow set bdepartures_]
      set bytesDbl [ns-int64todbl $bytes]
      set Kbytes [expr $bytesDbl / 1000 ]
      set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
    }
    set aux "$result $flowdata"
    set result $aux
  }

  # REGTCP Flows
  for {set fid 50} {$fid < [expr 50 + $regtcpflows]} {incr fid} {
    # extract data for this particular flow
    set fcl [$fmon classifier]; # flow classifier
    set flow [$fcl lookup auto 0 0 $fid]
    if {$flow != "" } {
      set bytes [$flow set bdepartures_]
      set bytesDbl [ns-int64todbl $bytes]
      set Kbytes [expr $bytesDbl / 1000 ]
      set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
    }
    set aux "$result $flowdata"
    set result $aux
  }


  set fid 100
  # extract data for this particular flow
  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]
    set Kbytes [expr $bytesDbl / 1000 ]
    set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
    set aux "$result $flowdata"
    set result $aux
  }


  set fid 120
  # extract data for this particular flow
  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]
    set Kbytes [expr $bytesDbl / 1000 ]
    set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
    set aux "$result $flowdata"
    set result $aux
  }


  set fid 140
  # extract data for this particular flow
  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]
    set Kbytes [expr $bytesDbl / 1000 ]
    set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
  }
  set aux "$result $flowdata"
  set result $aux

  set fid 160
  # extract data for this particular flow
  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]
    set Kbytes [expr $bytesDbl / 1000 ]
    set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
    set aux "$result $flowdata"
    set result $aux
  }



  set fid 220
  # extract data for this particular flow
  set fcl [$fmon classifier]; # flow classifier
  set flow [$fcl lookup auto 0 0 $fid]
  if {$flow != "" } {
    set bytes [$flow set bdepartures_]
    set bytesDbl [ns-int64todbl $bytes]
    set Kbytes [expr $bytesDbl / 1000 ]
    set flowdata "f$fid [$flow set parrivals_] [$flow set pdrops_] [$flow set pmarks_]
[$flow set pdepartures_] [format %.2f [expr $Kbytes * 100 / $possibleKBytes]]"
    set aux "$result $flowdata"
    set result $aux
  }


  # PERT Flows fid = 300
  set TOTparrivals    0
  set TOTpdrops       0
  set TOTpdepartures  0
  set TOTpmarks       0
  set linkutilz       0
  for {set fid 300} {$fid < [expr 300 + $nrpertflows]} {incr fid} {
    # extract data for this particular flow
    set fcl [$fmon classifier]; # flow classifier
    set flow [$fcl lookup auto 0 0 $fid]
    if {$flow != "" } {
      set bytes [$flow set bdepartures_]
      set bytesDbl [ns-int64todbl $bytes]
      set Kbytes [expr $bytesDbl / 1000 ]

      set aux [expr $linkutilz + [expr $Kbytes * 100 / $possibleKBytes] ]
      set linkutilz $aux

      set aux [ns-int64todbl [expr $TOTparrivals + [$flow set parrivals_] ] ]
      set TOTparrivals [ns-int64todbl $aux]

      set aux [ns-int64todbl [expr $TOTpdrops + [$flow set pdrops_] ] ]
      set TOTpdrops [ns-int64todbl $aux]

      set aux [ns-int64todbl [expr $TOTpdepartures + [$flow set pdepartures_] ] ]
      set TOTpdepartures [ns-int64todbl $aux]

    }
  }
  set flowdata "f300  $TOTparrivals $TOTpdrops $TOTpmarks $TOTpdepartures
[format %.2f $linkutilz]"
  set aux "$result $flowdata"
  set result $aux


  # total packets in the simulation
  set fid 999
  set bytes     [$qmon set bdepartures_]
  set bytesDbl  [ns-int64todbl $bytes]
  set Kbytes    [expr $bytesDbl / 1000 ]
    set flowdata "f$fid [ns-int64todbl "[$qmon set parrivals_]"] [$qmon set pdrops_]
[$qmon set pmarks_] [ns-int64todbl "[$qmon set pdepartures_]"] [format %.2f
[expr $Kbytes * 100 / $possibleKBytes]]"
  set aux "$result $flowdata"
  set result $aux


  puts $ffs "$result"
}

```