The Design and Simulation of a Transport Protocol for Interactive Network Applications

by

Richard James Wade

Submitted in accordance with the requirements
for the degree of Doctor of Philosophy.

The University of Leeds
School of Computer Studies

June 2000

The candidate confirms that the work submitted is his own and the appropriate credit has been given where reference has been made to the work of others.

Abstract

The Internet is currently an IP datagram network, which uses the Transmission Control Protocol (TCP) for guaranteed data delivery. In addition to providing a reliable data transport layer, TCP aids the stability of a large scale internetwork through congestion detection and avoidance algorithms.

Since TCP's inception in 1981, both the Internet, and the applications which use it, have evolved. The result is a broad spectrum of data traffic, being transported by protocols which were developed twenty, or more, years previously. Increasingly, the traffic being carried over the Internet is part of an interactive client/server session, established between hosts on widely separated networks. The number of router hops between such hosts means that an aggressive transport protocol for application data may attempt to send data, which exceeds the bottleneck capacity of a given network path. The result is packet loss which, for a guaranteed protocol, implies retransmission of data. Recent research has shown that current implementations of TCP, based on the original TCP algorithms, are inappropriate for the transportation of modern Internet traffic.

This thesis is concerned with the design, modelling, simulation, and study, of an experimental transport protocol. We aim to address the issues faced by current TCP implementations when transporting short, bursty, variable bit-rate, or bulk data in congested environments. In doing so, alternative methods of connection initialisation, flow control and congestion avoidance are implemented and simulated.

Through simulation with bulk, variable bit-rate and live HTTP trace data, we show how our experimental protocol is able to deliver data with successful throughput comparable with currently implementations of TCP. Due to its modified startup and congestion avoidance algorithms, however, it does so with significantly reduced packet loss and overall packet transmissions.

Acknowledgements

A number of people have aided the development of this work and thesis. Both Dr. Mourad Kara and Professor Peter Dew at the University of Leeds should be credited with this project's supervision, and have given significant input to its direction and publications. Furthermore, the members of the ATM-MM group at the University of Leeds, in particular, Dr. Karim Djemame, have provided invaluable feedback on my prototypes, mathematical and simulation models, and papers.

I would also like to thank the members of the Support team at the School of Computer Studies. Protocol development becomes very difficult without the ability to break networks and OS stacks in new and mysterious ways. Similarly, the development and prototyping of network protocols would not be possible without solid simulation tools. For these, I would like to thank the developers involved in the NS and REAL projects.

Torch Telecom were also instrumental in providing additional funding for my CASE studentship through the EPSRC.

Finally, I would like to dedicate this work to my parents, who have given tremendous support throughout my academic career. Without them, none of this would have been possible. Thank you.

Declarations

Some parts of the work presented in this thesis have been published in the following articles:

Wade, Kara, Dew, "Proposed Modifications to TCP Congestion Control for High Bandwidth and Local Area Networks", *6th IEEE Conference on Telecommunications*, (July 1998).

Wade, Kara, Dew, "Study of a Transport Protocol Employing Bottleneck Probing and Token Bucket Flow Control", *IEEE International Symposium on Computer Communications*, (July 2000).

Wade, Kara, Dew, "Modeling and Simulation of STTP, a Proactive Transport Protocol", (pages 486–486), *IEEE International Conference on Networking*, (September 2000).

Contents

List of Figures

List of Tables

Acronyms

ACK	Acknowledgement packet
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
ATM	Asynchronous Transfer Mode
BSD	Berkeley Software Distribution
CGI	Common Gateway Interface
DARPA	Defence Advanced Research Projects Agency
ECN	Explicit Congestion Notification
EPD	Early Packet Discard
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISP	Internet Service Provider
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
MTU	Maximum Transfer Unit
NEST	Network Simulation Testbed
NS	Network Simulator
OSI	Open Systems Interconnection
RED	Random Early Discard
RFC	Request for Comments
RSVP	Resource Reservation Protocol
RTP	Realtime Transport Protocol
RTCP	Realtime Control Protocol
RTSP	Realtime Streaming Protocol
RTT	Round Trip Time

SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TOS	Type of Service
UDP	User Datagram Protocol
UUCP	Unix to Unix Copy Protocol
VPN	Virtual Private Network
WWW	World Wide Web

Chapter 1

Introduction and Background

The issue of transport protocol design has become an important factor in the future of the Internet with the proliferation of multimedia and interactive applications. The majority of Internet applications are client/server in nature and therefore require communication between two, potentially widespread, network hosts.

The number of hosts connected to the Internet has grown at an unprecedented rate, and its penetration into all corners of the globe has brought a wide variety of network quality and capacity. In developed nations, rapid expansion has taken place both for domestic and business users who, at the time of writing, generally connect at between 28.8kb/s and 128kb/s. Larger businesses with leased lines to their Internet Service Provider (ISP), may work with T1 (1.55Mb/s) connections and higher. Government or educational institutions generally have even higher capacity links up to T3 (45Mb/s).

When a connection is initialised between two hosts, little is known about the physical network's capacity or reliability. Even though both hosts may be located on high-capacity local area networks, with high performance connections to the Internet, the call may be routed via highly congested or unstable portions of the public network. This implies that the *bottleneck* for this connection is unknown, unpredictable and not under the control of local administration. It is therefore unwise for an application to request or send data at a pre-determined rate.

1.1 The Transmission Control Protocol

The Internet is currently an IP-based network which runs TCP [41] (Transmission Control Protocol) or UDP [40] (User Datagram Protocol) at the transport layer. For guaranteed data services, TCP is the recommended protocol as it provides windowed flow control and retransmission of lost/corrupt data. Furthermore, a TCP stream will modify its transmission rate according to current network congestion. Conversely, UDP does not provide any flow control or congestion avoidance facilities, and is defined as an "unreliable" transport layer. UDP has been the source of much contention over recent years due to multimedia network applications such as voice and video streaming, flooding the Internet with unresponsive protocol streams. Applications which use UDP may not incorporate TCP Friendly [44] congestion avoidance mechanisms, thus affecting the quality of service available to competing data streams.

TCP initialises a connection with an algorithm known as *slow start*. The rate at which TCP transmits traffic into a network is governed by the size of its *congestion window* (cwnd), which is normally an integer value, representing the number of segments currently allowed to be in transit on a given connection. By initialising this value to 1, and incrementing it each time an acknowledgement (ACK) is received from the destination node, TCP can achieve a self-clocked method of bandwidth discovery. In practice, increments to the congestion window follow an exponential curve and continue until either data is lost, or a pre-determined limit is reached. This limit is known as the *slow start threshold* (ssthresh) and is set to 65535 bytes or the receiving host's advertised window size.

The ssthresh mechanism prevents the sender from overflowing network buffers during slow start. In practice, however, this threshold could be higher than the bottleneck capacity of the intervening network. Once the congestion window value reaches that of ssthresh, TCP enters its congestion avoidance phase. This involves incrementing the cwnd with every ACK that is received, therefore a linear process. Action is only taken if data is lost or three ACK's with the same sequence number are received. This implies that the destination node is receiving out of order packets, or that data has been lost. In this situation, TCP decreases its congestion window by 50% and re-enters congestion avoidance. If a timeout should occur, i.e. the time taken for the destination host to reply with an ACK packet exceeds a given limit, then the congestion window is set to 1 and TCP re-enters slow start.

Conversely, UDP does not support such flow control or error handling. It falls to the application developer to provide these features in a custom transport layer, should the be required. The concern is that if such software is badly implemented, it may be unfair towards *TCP Friendly* connections which share the same network links.

1.1.1 A History of TCP

1.1.1.1 1988

Jacobson and Karels' paper [21] on TCP congestion avoidance 4.3BSD Tahoe was released, which utilises:

- Slow Start
- Congestion Avoidance
- Fast Retransmit

The Slow Start and Congestion Avoidance algorithms were documented in [21], whereas Fast Retransmission facilities were not ratified until 1997 in RFC 2001 [1] [48]. Slow Start and Congestion avoidance were the result of a "congestion collapse", which took place in 1988 on the DARPA network at Berkeley. This added facilities to TCP, which gave it the ability to detect and avoid severe network congestion. If TCP were able to slow down its data transmission rate when congestion is detected, then massive packet loss will be prevented, and overall performance improved. Similarly, Fast Retransmission of data allows TCP to retransmit data which it suspects has

[1] RFC 2001 describes in detail TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. Its aim is to document and standardise current implementations.

been lost in transit due to the receipt of duplicate acknowledgement packets for a previous piece of data.

1.1.1.2 1990

The Reno release of 4.3BSD changed its TCP implementation only slightly. The facility which was added permitted TCP to increase its transmission (congestion) window by the number of duplicate acknowledgements it had received after retransmission. This algorithm accelerates recovery of a stream's transmission rate after packet loss. A further enhancement of TCP for the 4.3BSD Reno release was to allow TCP to only halve its current transmission rate in case of Fast Retransmission (detailed in 4.3BSD Tahoe). Prior to this extension, each packet loss caused TCP to enter the Slow Start algorithm, and reset its congestion window to a single segment.

1.1.1.3 1993

Lawrence Brakmo et al. at the University of Arizona, extended the Slow Start and congestion avoidance algorithms of TCP Reno. They implemented their new algorithm, called TCP Vegas, on an experimental platform called the X-Kernel, and published a performance study of Vegas in [11], at SIGCOMM 94. The modifications to Slow Start include a more cautious expansion of the congestion window during connection initialisation. This aims to prevent packet loss due to aggressive transmission rates. Similarly the congestion avoidance algorithm was improved to sense network congestion, and adjust its transmission rate accordingly. This was achieved by obtaining an expected throughput figure for a given time period, and comparing this with the actual connection statistics. If there is a notable disparity between these values, then the transmission rate will be adjusted. [11] includes performance testing, which shows 40-70% performance improvement over TCP Reno in Brakmo's simulation experiments.

1.1.1.4 1996

RFC 2018 [2] [33] details extensions to TCP Reno in the form of Selective Acknowledgement Options. This enables a TCP receiver to overcome the limitation of cumulative acknowledgements in TCP Reno. With cumulative packet acknowledgements, a TCP sender can only learn about a single lost packet in each round trip time. With Selective Acknowledgement options (SACK), a receiver can acknowledge data as it is received. This enables the sender to only retransmit that which has been lost, and not packets which have already been transported successfully.

1.1.1.5 1997

RFC 1122 [3] [8] detailed the basic requirements for a TCP implementation, that it should use a Slow Start mechanism, and congestion avoidance to prevent network overload. RFC 2001 [48] consolidated RFC 1122 with work done by Jacobson and Karels. Furthermore, it documented the Fast Retransmit, and Fast Recovery algorithms for the Internet community.

1.2 Interactive Network Applications

While *streaming* applications such as voice over IP, audio or video streams may use UDP for their transport, the majority of Internet applications use TCP for guaranteed data delivery. The most widespread of these is without doubt the World Wide Web, which is a client-server application, used to present information and data through interactive Hypertext documents, or Web pages. These are transmitted using the HyperText Transfer Protocol (HTTP) [17], which in turn uses TCP.

Under normal conditions, a Web client will make a HTTP request for a given page. The server then initialises a separate TCP connection for each page component

[2] RFC 2018 describes how TCP Selective Acknowledgement Options can improve the performance of TCP conversations with bursty packet loss.

[3] RFC 1122 documents the IP and TCP functionality required for Internet-connected nodes (hosts).

back to the client. However, work has been conducted into Persistent HTTP [17] which aims to multiplex these connections together in order to form a single, stronger TCP connection.

The nature of the Web is such that a user may make rapid page selections in order to obtain specific information. Although a great amount of work has been done to optimise Web client and server efficiency, the underlying transport protocols can still be considered a factor in user-perceived latency. While HTTP itself has undergone several revisions in recent years, the components which make up Web documents are also evolving at an alarming rate. The result is that Web server and document technologies are now able to present an increasingly interactive interface to the user with a wider variety of components being presented to the client software. Furthermore, HTTP itself is under pressure to evolve in order to transport such components efficiently and in a manner which is conducive to the interface it aims to provide.

1.3 Internet Services

While the World Wide Web Consortium (W3C) revises the HTTP specification, it falls to the Internet Engineering Task Force (IETF) to improve common Internet protocols and provide the necessary functionality, performance and flexibility to the application layer. For example, the Internet Protocol version 6 (IPv6) has recently been specified in order to accommodate future address space requirements as well as extended support for differentiated levels of network service.

The Architecture for Differentiated Services (DiffServ) allows a network operator to provide certain guarantees to their customers in terms of Quality of Service (QoS). In more detail, traffic can be classified according to certain bounds given by its Type of Service (TOS) specification. This TOS header, present in IPv4 and IPv6 packets, can be used to describe acceptable bounds for queueing delay, jitter, or packet loss.

It is therefore anticipated that differentiated services will proliferate with customers subscribing to ”Bronze”, ”Silver” and ”Gold” levels of traffic provision.

Bronze customers may simply receive the current standard of "best effort" with no performance guarantees for their traffic. Subscribers to the Silver category may have their traffic prioritised over the Bronze customers according to a subset of the QoS parameters. Gold customers may be able to specify all available QoS parameters while paying a premium fee for their traffic.

Within the realms of a QoS-oriented, or Diffserv environment, the role of the IP transport protocol is diminished somewhat as the secondary network services take on a portion of its functionality. Whereas in a mixed media, best effort network such as the current Internet, the transport protocol aims to contend with varying delay, packet loss and cross traffic, a Diffserv network is able to guarantee at least a subset of these parameters. Therefore, the requirements for next generation TCP are somewhat different to those when it was specified in 1981 for DARPA's RFC 793 [4] [41].

1.4 TCP Issues and Alternative Implementations

Naming conventions for TCP are traditionally based around the BSD revision in which they appear. Both of the following appear in releases of 4.3 BSD Unix. First, there was TCP Tahoe which implemented RFC 793 in the BSD kernel in addition to slow start and congestion avoidance algorithms from RFC 2001. TCP Tahoe is detailed in Jacobson and Karels' paper on Congestion Avoidance and Control [21]. This protocol was superceded by TCP Reno, which adds fast retransmit and fast recovery from RFC 2001 [48], and selective acknowledgements from RFC 2018. TCP Reno is currently the most common implementation used in network operating systems and as such, has been studied in great detail. The majority of performance comparisons and models in this work will use TCP Reno as their benchmark protocol.

In 1994, Brakmo et al at the University of Arizona published [11], which detailed new algorithms for TCP slow start and congestion avoidance. In particular, TCP

[4] RFC 793 describes the initial Transmission Control Protocol for the ARPA network.

Vegas claims "between 40% and 70% improvement" in performance over TCP Reno with around one fifth the number of packet retransmissions. This performance increased was achieved through better utilisation of available bandwidth by deploying proactive congestion avoidance algorithms. An alternative, lightweight implementation of this solution to congestion avoidance is used in our experimental protocol later in this work.

One fundamental criticism of TCP Reno and its variations, is that the core algorithms rely on packet loss in order to detect network congestion. In essence, they use a linear increase, multiplicative decrease algorithm which, despite having been shown to exhibit *fair* behaviour between competing streams is both lossy and inefficient when compared with proactive congestion avoidance [11].

In particular, TCP Reno has been observed to lose the majority of its data while in slow start which, in comparison with the algorithms used by TCP Vegas, are aggressive. The problem arises when, in slow start, TCP Reno doubles its transmission rate every RTT. It continues to do so until it reaches the slow start threshold, the remote hosts's advertised window size, or loses data. In a highly congested network, particularly one with a high bandwidth delay product, this means that when losses do occur, they are likely to be in the order of half the current congestion window. Given that TCP is a guaranteed delivery protocol, this data has be retransmitted and if losses have occurred, this will be at a much lower rate (50%) than before. This is due to TCP's congestion avoidance algorithm, which will halve its transmission rate upon data loss. TCP then enters its congestion avoidance algorithm.

Fast retransmission and recovery are defined in RFC 2001 [48]. Fast retransmission of data is triggered in TCP Reno by the source host receiving three duplicate acknowledgements for data it has sent. The advantage of this mechanism is that it does not have to wait for a timer to expire (a timeout) before retransmitting potentially lost data. The receipt of three duplicate ACK's at the source means that the remote host has received three packets which contained out of order sequence numbers. Occasionally, packets may be re-routed in the Internet, so certain data may arrive out of order and have to be re-ordered before being passed to the application.

Once fast retransmission has been performed, TCP Reno uses fast recovery to maintain its state in congestion avoidance. This means that unless a retransmission is performed due to packet timeout, a slow start is not performed. If a timeout does occur, ssthresh is set to one half of the current congestion window, the congestion window is set to a single segment, and slow start ensues. In fast recovery, however, the congestion window is set to the value of ssthresh as opposed to a single segment.

1.5 Motivation

The issue of flow control in modern computer networks has been highlighted in recent years with a massive increase in the amount of bursty Internet traffic. The proliferation of the World Wide Web (WWW) has lead to a desire for short, high-bandwidth connections in order to transfer relatively small documents, images and program code in the least time possible.

This is in contrast with traditional Internet applications such as FTP, which generally consisted of longer duration connections. Furthermore, the relative *fairness* of bandwidth allocation was considered more important than timely delivery. Such lengthy conversations meant that the precise dynamics of individual connections were not of primary concern when, for example, slow start only formed a small percentage of the total duration.

Given that the majority of current WWW pages contain a small amount of text content, this can be transferred in a handful of TCP segments. In contrast, a lengthy FTP connection may transfer hundreds or thousands of kilobytes and take several minutes to complete. During this time, a strong, established TCP stream may be formed and its bursty profile reduced compared with that of WWW traffic. While TCP's slow start and congestion avoidance algorithms are capable of efficiently transporting lengthy data transfers, modern Internet applications no longer fit this profile.

It is therefore of paramount importance to ameliorate the perceived performance of WWW traffic through improvements to the transport layer. While HTTP is constantly being revised by the W3C, one way to increase data throughput in real terms, is by optimisation and tuning of the Internet's transport protocol.

Modern Internet applications set the criteria by which this performance will be measured and allow us to derive suitable algorithms for their solution. We are therefore faced with a set of user requirements, namely to deliver short, bursty transfers of data in as short a time as possible.

1.6 Research Context

In our experience with simulation experiments [49] [51] [50], TCP has not shown itself to be suitable for realtime, interactive, network applications. In particular, those involving bursty, multimedia data exchange. The reason for this is that due to its slow start algorithm, TCP is not able to quickly establish a connection which fully utilises the available bandwidth. In fact if a network is highly congested, a data exchange may take much longer than desired due to packet retransmission, which is likely to occur during aggressive expansion of the congestion window.

The arrival of more intelligent network services such as Diffserv [36] and RSVP [9], which allow bandwidth allocation in addition to bounded QoS on a per connection basis, means that TCP is now operating in an environment which is very different to that for which it was designed. The creation of Virtual Private Networks (VPN's) using such technologies make it increasingly unlikely that data will be re-routed, unless some sort of technical problem occurs. Furthermore, connections between nodes on such a network may have a given bandwidth allocation, which makes slow start merely an inefficient use of the available resources.

In addition to advances in networking technologies, the physical layer upon which TCP/IP operates, is now substantially more reliable than, ten years ago. Data loss or corruption due to physical error is now only likely every 10^{-6} packets. The vast majority of data loss is caused by network congestion and overflowing queues at the router or switch. The major concern has therefore shifted from an unreliable network to one which is reliable, but highly and unpredictably congested.

The objectives for TCPng are therefore to be more concerned with queue management and efficient retransmission, connection and recovery. However, much of the network congestion in effect on the Internet will be cross traffic streams which may, or may not be TCP friendly. The design of a next generation transport protocol must take into account the diversity of traffic on modern networks and behave fairly towards other users.

There are two main definitions of fairness for transport protocols, basic max-min fairness states that a protocol should make maximal usage of available network resources, but be able to share equally such resources amongst data streams. Given n connections operating on a given link, a state of equality would exist if the throughput for each connection was an equal share of the total link capacity. If any connection were to increase its throughput, it would be to the disadvantage of other streams in this situation.

Alternatively, there is proportional fairness which aims to maximise $\Sigma_R \log \lambda_r$ according to a link's resources, where λ_r is the rate allocated to a given TCP connection. Proportional fairness is a congestion control scheme in which routing priority is assigned to a given stream according to the quality of service assigned to, or purchased by, the user. In proportional fairness, every network resource has a "price" associated by the network administration. Users of the network are allocated network resources according to how much they are able to "pay". The price allocated to each resource is adjusted in real time, according to the current state of resource demand e.g. if demand is high, then the price for a given resource will increase. The detailed process and mathematical modelling can be found in [24], and [32].

In addition to fair and considerate behaviour, a next generation transport protocol should be able to make full use of available network resources. In a guaranteed QoS network, this may be reserved or allocated bandwidth. Given that such reservations may be charged at a significantly higher rate than best effort services, it is important for data to be exchanged in the most efficient manner.

1.7 Objectives

The work presented in this thesis focusses on optimisations to the transport layer, which yield greater performance in terms of successfully transmitted data. Our aim was to minimise lost data, while maximising overall throughput.

The aim of such optimisation is to update existing TCP protocols in order for them to function in accordance with forthcoming network services. Furthermore, technologies which already exist on the physical layer mean that design concerns are somewhat different to those ten years ago. In summary, while certain assumptions can now be made with regard to modern networks, there are some very different concerns with regard to congestion avoidance and control, particularly with regard to real time and multimedia applications.

In order to address the issues faced by TCP and next generation transport protocols, this work will address several key areas:

[K1] the survey of existing protocol research in order to identify key problem areas

[K2] the evaluation of TCP modifications as a potential solution

[K3] the design and simulation of an experimental protocol, which deploys proven techniques in bandwidth discovery, flow control, and congestion avoidance

    [3a] Connection initialisation and startup

    [3b] Congestion avoidance and control

    [3c] Packet loss and recovery

[K4] evaluation of the experimental protocol against existing TCP implementations with a variety of network traffic models

Original contribution is made through modifications to the existing TCP Reno implementation (referred to as C1) and with an experimental protocol (C2), which has been designed specifically to address the issues mentioned above. C1 is addressed in section 3.1, and C2 throughout chapter 3. Simulation of the resulting protocol is carried out in chapter 4 (C3). Further detailed analysis of our protocol simulations can be found in chapter 5 (C4).

Chapter 2 discusses work which is closely related to that covered in this thesis, namely publications on TCP, related protocols and congestion avoidance algorithms. In order to address K1, variations on TCP's congestion avoidance and slow start algorithms are discussed, with reference to published work in this area. The body of this work is presented in chapters 3 through 4, where our experimental protocol, STTP, is described, tested and evaluated. The acronym, STTP, stands for Shaped Token Transport Protocol. K2, K3 and K4 are consolidated in these chapters through simulation experiments. The rationale and design of STTP is covered in chapter 3, followed by performance testing and results in chapter 4. Chapter 3 addresses K3 by describing the algorithms used in STTP to address deficiencies in TCP when transporting modern Internet traffic. Further discussion of the simulation results takes place in chapter 5 with conclusions and future work in chapter 6. K4 is discussed further in chapter 5, where simulation results from both NS and REAL are analysed, and their performance compared.

Chapter 2

Related Work

2.1 Traditional TCP

Jacobson and Karels' key paper of 1988 [21] describes how, in response to a congestion collapse of the Internet in 1986, congestion avoidance and control algorithms were incorporated into the transport control protocol (TCP) of the time.

They describe a "self-clocking" protocol which uses windowed flow control and packet acknowledgements to gradually increase the flow of data into a network. This continues to the point at which congestion occurs and data is lost, at which time multiplicative decrease algorithms are used to reduce network load. This approach to congestion control is central to TCP's core algorithms and is operational in both TCP Reno and TCP Tahoe, the most common active implementations of TCP.

The design of RFC 793 TCP (from the Jacobson and Karels' paper) is based on the concept of "packet conservation" with TCP streams in "equilibrium". Equilibrium is termed as the steady state in which a new packet is not injected into the network until one has arrived at the destination node. In order to achieve equilibrium, a new slow start algorithm for connection startup was designed.

Slow start was created in order to kickstart the self-clocking TCP algorithms. As TCP depends on the receipt of acknowledgement (ACK) packets to trigger the release of new data into the network, there had to be some way of increasing the flow until it reached equilibrium. Therefore, Jacobson and Karels' introduced a new slow start state to TCP. While in this state, TCP increments its transmission window (congestion window, cwnd) for every ACK that it receives. To reach a given window size W from slow start, RFC 793 TCP takes $\log_2 W$ round trips. This process gives TCP the properties of a self-clocking protocol which will regulate its transmission according to the available network resources. During slow start, TCP quickly increments its transmission rate as data is received and acknowledged by the remote host. Such behaviour aims to reduce packet loss during connection initialisation.

The TCP then sends the minimum of the congestion window and the receiver's advertised window (buffer) size. This mechanism takes into account the wide variety and specification of machines on the Internet by not allowing a powerful host to overload a less powerful one with floods of data. In this manner, Jacobson and Karels' TCP was able to resolve the issues of congestion on the Internet of the time. A receiver's advertised window size indicates the number of bytes available for data at the receiving end of a connection. Therefore, the transmitting side should not allow more than this amount of data to be unacknowledged at any time. To do so would give a high risk of packet loss due to buffer overflow at the remote host.

In this section, we have covered the fundamental concepts of TCP, that it is a self-clocking protocol, which guarantees data delivery and which will react to network congestion in order to minimise packet loss. We continue with the examination of alternative TCP implementations. An alternative implementation is a transmission control protocol which has been designed with clear and distinct criteria. Alternative implementations are usually completely redesigned protocols, which derive few algorithms from standard TCP. Conversely, we will also discuss TCP variants. A variant is an improvement on existing TCP which aims to solve specific problem areas, or to improve performance under certain conditions.

2.2 Alternative Implementations

The rapid expansion of the Internet (from the mid 1990s), both physically and in terms of traffic types has prompted a great deal of academic and industrial research. In particular, the issue of streaming and interactive, multimedia traffic using UDP and proprietary transport layers.

2.2.1 Real-Time Protocols

One such highly successful project was that of RTP, the Realtime Transport Protocol. RTP is an IP-based protocol which supports realtime, multicast and unicast, audio and video streams. The essence of this work is published in RFC 1889 [45], which specifies RTP and RTCP, the framework's two main components.

RTP provides timestamping, sequence numbering, source identification and payload format information. These fields can be used by multimedia applications to ensure in-order, regulated playback of audio or video streams. Furthermore, it is possible to combine such streams and to synchronise their output using RTP's timestamp.

RTCP is the Real-Time Control Protocol, a feedback mechanism for Quality of Service (QoS) applications. In conjunction with RTP, this provides facilities for applications to exchange administrative information, such as the monitoring of network resources.

RTP was adopted by Netscape in 1996 for use in their "Netscape LiveMedia" audio/video streaming application. This led to further development in conjunction with Columbia University and RealNetworks to produce RTSP [46]. RTSP, the Real-Time Streaming Protocol, works in conjunction with RTP and RTCP to provide a simple means of accessing remote multimedia services. In many cases, RTSP is used in conjunction with HTTP to allow clients simple Web access to stored multimedia streams.

The RTP suite provides a great deal of useful functionality to multimedia applications, but relies on them to take appropriate action on receiving congestion notification. Furthermore, it does not explicitly provide any QoS facilities but depends on lower network services, such as RSVP [9], to allocate resources. It is therefore most suited to ATM or QoS-enabled IP environments where bandwidth and resource allocations can be controlled. However, both RealNetworks and Netscape have shown that it can be used effectively in a best-effort Internetwork.

Table 2.1: Summary of Protocol Traffic Support

Protocol	Bulk Transfer	Bursty Traffic	Multimedia Traffic	Interactive Multimedia Traffic
TCP Reno	X	X	-	-
TCP Vegas	X	X	-	-
Dual/Tri-S	X	X	-	-
Packet Pair	X	X	-	-
RTP	-	X	X	X

Table 2.1 shows the transport protocols covered in this chapter, and the traffic types to which they are suited. Those based on traditional TCP (Reno, Vegas and Dual/Tri-S) are most suited to bulk transfer (FTP, SMTP, NNTP, etc.). Packet Pair potentially provides greater support for bursty traffic (HTTP, Telnet) due to its repeated estimation of the current network state using packet pair probes. However, only RTP provides true support for (Interactive) Multimedia traffic. Features such as stream synchronisation are of great importance, particularly for joint viewing or interactive sessions.

In TCP's lifetime, many propositions have been made to subtly alter the behaviour of certain algorithms. In particular, slow start has been the focus of much attention. A *variant* of TCP is a protocol which is able to provide transport layer functionality, while exhibiting significantly different behaviour to standard TCP. Variants of TCP often include experimental, or alternative, startup methods and congestion avoidance algorithms. Section 2.2.2 will discuss variants of standard TCP. Conversely, a *TCP modification* is an alteration to standard TCP algorithms in order to improve performance under certain network conditions. TCP modifications are discussed in section 2.2.3.

2.2.2 TCP Variants

While earlier TCP implementations used packet loss or timeout as their only indication of network congestion, a great deal of work has been done to examine alternative methods of detecting resource availability. Given that TCP is an acknowledgement based protocol, the Round Trip Time (RTT) measurement taken whenever an ACK is received, has proven most useful when determining the state of the intervening network [3] [38]. Increases in RTT, or drops in throughput over a given time period, can indicate an increase in network cross traffic. This leads to increases in router queue length on a given path, thus affecting a connection's end-to-end latency.

Slow Start and Search, Tri-S [52], was proposed by Zheng Wang and Jon Crowcroft and uses variations in RTT to detect possible network congestion. A normalised throughput gradient is calculated, which represents the projected throughput for a connection with a given congestion window size. Should the gradient fall below a pre-defined threshold, the transmission rate is decreased. However, if an increase in throughput is not making a significant impact on the perceived network load, then the window is increased at TCP's standard, linear, 1/cwnd rate.

In practice, Tri-S's congestion avoidance exhibits similar behaviour to other proactive systems. Its algorithms aim to reduce delay, and therefore minimise router queues. The implication of this is that router buffers suffer less congestion, therefore drop fewer packets. This property of a proactive system is a side effect of sensing congestion by variation in RTT, but is dependent on threshold settings within the protocol.

One potential hazard of using pre-defined thresholds is that a given value may not be applicable to a wide variety of bandwidths or RTT's. For example, threshold values which yield the desired effect on a 10Mb/s LAN are unlikely to function in the same manner on a high latency satellite connection. This is due to greatly increased latency, which affects the behaviour of ACK-based protocols such as TCP.

A similar solution to Tri-S, the DUAL algorithm, was proposed by the same research group in [53]. DUAL uses traditional TCP timeouts to detect heavy network congestion, in which case it reduces the slow start threshold to 50% of its current value, and the congestion window to a single segment. Under normal congestion avoidance, it compares each RTT measurement and, if necessary, will make adjustments to its transmission rate every other round trip. If the RTT is sensed to be increasing, the current transmission rate is reduced. The reason for adjusting the transmission window every other round trip is to prevent rapid fluctuation in the rate of transmission, and to allow a more smooth estimation of network congestion to be made due to an increased number of data samples.

This mechanism is achieved by maintaining RTT_{min} and RTT_{max} variables, which are initialised to a very large integer (out of the possible range for this application) and zero respectively. When a new RTT measurement is obtained these variables are updated accordingly to store the minimum and maximum RTT's experienced on the current connection. If the new RTT value is greater than $(RTT_{min} + RTT_{max})/2$, then the current window is adjusted to 7/8 of its current value. During the course of Wang and Crowcroft's experimentation and simulation, 7/8 of the current window size was shown to give "the best" compromise between performance and fairness to other data streams. We discuss the compromise between performance and fairness in section 5.2.3, chapter 5.

The aim of the DUAL algorithm is to reduce oscillation of TCP's window size after slow start. This phase is where TCP Reno relies on packet loss or timeout to detect network congestion. Upon doing so, the window size is either reduced by 50%, or to a single segment, depending on the event. By detecting increases in RTT, DUAL is able to reduce its transmission rate before such losses occur. This avoids unnecessary timeouts or packet loss, hence reducing the amount of oscillation and variance in the transmission window.

TCP Vegas [11] is one variant in which the authors claim specific, quantitative performance improvements over TCP Reno. A 40% improvement in throughput coupled with one fifth to one half the number of packet drops is cited in [11]. TCP Vegas's congestion avoidance algorithms compare the actual measured throughput with an expected value. The expected throughput value is obtained using the current measured RTT and congestion window size. Comparing this with the measured throughput for the following RTT will give an indication of the current state of network congestion. This technique is also used in TCP Vegas's modified slow start algorithm, which only increases the congestion window every other round trip. This allows throughput measurements to be taken and compared with the expected value for that size window. Should congestion be experienced during slow start, TCP Vegas can move into the congestion avoidance phase before packet loss occurs. It is noted by the authors that packet loss during slow start is common among TCP Reno implementations. Should congestion be encountered during the startup phase, the aggressive nature of this algorithm loses around 50% of the current transmission window's packets. Short, bursty, data transfers, such as Web page component downloads, will often spend 100% of their time in TCP's slow start algorithm. If TCP is tranferring 1.5kilobytes in each packet, a single Web page component of less than 10kilobytes will be completed in around six packet transmissions. For TCP, this is only three round trips. A more cautious slow start algorithm can greatly reduce the number of packets lost and retransmitted during the lifetime of a short TCP connection.

Vegas measures the actual throughput for a connection as the number of bytes transmitted in a given measured RTT. In a similar manner to DUAL, it also maintains a record of the minimum RTT (BaseRTT) experienced on the current connection. The expected throughput can then be calculated using WindowSize divided by BaseRTT. In congestion avoidance, the difference between the expected and actual throughput is taken, and compared with two threshold values, α and β where $\alpha < \beta$.

If the difference lies between α and β, no action is taken. If the difference is below α, then linear increase takes place. If it is above β, then linear decrease is enforced. In [11], several experiments were performed with different values for both α and β and showed differing levels of throughput, timeouts and packet retransmission. The effect of increasing the difference between α and β was to make the protocol less sensitive to variation in network congestion. In order for a change to be made to TCP's transmission rate, it has to fall outside either bound. If a protocol becomes insensitive to the current network state, then it is unlikely to take advantage of free bandwidth, or to reduce its transmission rate when congestion increases. Brakmo et. al have published simulation results which demonstrate this technique in [11].

In her work with Explicit Congestion Notification (ECN) [18], and Random Early Detection (RED) [19], Floyd conducted experiments which highlighted the benefits of alternative congestion avoidance algorithms. In [18], TCP Reno was modified to respond to ECN signals and is simulated on networks with routers employing RED algorithms. RED employs a similar technique to that of Jain's DECbit [42], in which router queue sizes are monitored and a congestion bit set should they exceed a given limit. When marked packets are received at the source, TCP can then act on this congestion feedback with multiplicative decrease in its transmission (congestion) window.

The benefits of RED and ECN are comparable with that of pro-active congestion avoidance algorithms. Action is taken by the transport layer prior to packet loss on a per-connection basis. The result is a decrease in unnecessary packet drops and timely reaction to the current network state. Such benefits are quantified in our experiments with STTP in chapter 4. Our simulation results show that a transport protocol with pro-active congestion avoidance is able to avoid packet loss, and achieve packet goodput comparable with traditional TCP, while transmitting significantly fewer packets.

2.2.3 TCP Modifications

In contrast to variant protocols such as TCP Vegas and alternative congestion avoidance algorithms such as DUAL, Tri-S etc., there have been many attempts to improve TCP's performance with minor modifications. The most significant of these were fast retransmit and recovery, described in RFC 2001 [48] and obsoleted by the proposed standards in RFC 2581 [4]. In a note sent to the *end2end* mailing list by Jacobson [22], he details modifications to TCP's congestion avoidance algorithm. These modifications form the basis for TCP's fast retransmit and recovery mechanisms. RFC 2581 clarifies and consolidates many proposed changes to TCP. It re-specifies the initial window size of a connection to be two TCP segments instead of one. This modification will double the initial data transfer between two hosts, thus aiding the exchange of short bursts of data such as Web page components. The restart behaviour of an idle TCP connection is also re-defined in RFC 2581, as having to perform a slow start from the standard initial window size of two segments. The generation of ACK packets is also specified as adhering to the specification laid out in RFC 1122 [8]. A TCP receiver should generate acknowledgements for at least every other full size TCP segment that it receives. Furthermore, ACK's should be delayed by at most 500ms.

While such modifications can significantly improve the performance of existing TCP implementations, the 1999 standards still adhere to traditional slow start and duplicate acknowledgement/packet loss congestion indicators. RFC 2581 does refer to experimental slow start mechanisms, such as those outlined in RFC 2414 [37]. Despite work in this area having been published, precise details were not included in RFC 2581. Furthermore, the work done on TCP Vegas and other proactive congestion avoidance algorithms does not seem to have influenced current specifications for Internet transport layers.

Work conducted by Allman, Hayes and Ostermann [2] examines the feasibility of the above slow start modifications. An increase in TCP's initial window size to four segments was shown to yield an 80% increase in throughput for short connections using HTTP-like traffic, compared with standard TCP Reno. However, a side effect of this more aggressive algorithm is a slight increase in the packet drop rate, of 0.1 segments per transfer. When used in conjunction with the new recommendations for ACK generation, detailed in [4], a 150% increase in throughput and one segment per transfer in packets dropped was noted.

Further study in this area was conducted in [39], where further experiments were carried out using the NS simulator [5]. Their conclusion was that an increased window size at TCP startup helped improve perceived TCP performance. In particular, short data transfers will complete more quickly, due to a larger initial window size. The matter of packet loss over low bandwidth connections was studied more closely by Shepard and Partridge in [47]. In their experiments, also with the NS simulator, they discovered that an increased TCP window size of four segments was not detrimental to the individual connection. Their results showed that a four-segment initial window size showed up to 30% performance improvement over the standard single-segment value. These results are discussed in RFC 2414 [37], in addition to discussing further related work. At this time, however, TCP implementors are reluctant to impose a four segment slow start on the Internet. The reason for this is that further study needs to be made of the potential impact to very large scale network performance. The body of this thesis is focussed on the design of a new transport protocol, and not on the modification of existing TCP algorithms. The reason for this decision is discussed in chapter 3.

An alternative method of connection startup is addressed by Keshav in [26]. In this work, the *Packet-Pair Probe* is proposed as a way of estimating the bottleneck bandwidth of a given connection. Its premise is that the delay introduced when packets are forwarded over the bottleneck link will be preserved and can be used to discover the lowest link speed on a given network path. By transmitting packets back-to-back, i.e. with no inter-packet delay, their spacing at the remote host can be analysed to give a good indication of the current path state. This method will be more closely analysed in chapter 3, where it is incorporated as part of our protocol design.

Further research has been done in this area by several groups. The Packet-Pair Probe was used by Hoe in [20] to estimate a connection's bottleneck bandwidth. The probed value was then used to calculate the bandwidth-delay product, the byte equivalent of which is taken to initialise the connection's slow start threshold (ssthresh). Performance improvements were seen when employing the probed value over standard, default TCP settings. In particular packet loss for short connections, such as HTTP requests, was reduced. This reduction in packet loss also improved overall performance by eliminating unnecessary timeouts during the startup period.

In [38], Paxson discusses the Packet-Pair in great detail, and attempts to resolve many of its shortcomings through use of Packet-Pair *bunches* and receiver-side bottleneck estimation. By using a number of packets in succession, as opposed to only two, the likelihood of packet loss is greatly reduced. Furthermore, a more accurate estimation of the bottleneck link can be taken using multiple probe values. In [38], Paxson considers the possibility of multiple channels and routes for a TCP connection and concluded that by using "Packet Bunch Modes" (multiple probe packets) and receiver-side calculations, the issue of multiple routes, load-balanced connections, and bottleneck changes can be resolved.

Further work in this area was conducted by Allman and Paxson in [3], where Packet Bunch Mode algorithms were run on large network trace data sets of over 11,000 connections. They concluded that using Packet Bunch Modes, in conjunction with receiver-side bottleneck estimation, provides distinct benefits to over 25% of connections in their experiments.

A similar approach was used by Ahlgren et al [1] where chains of one hundred packets were transmitted between hosts in Uppsala (Sweden), Massachusetts (USA) and Cambridge (England). Their results correspond with those of Carter and Crovella [12] and show stable estimations with trains of ten to fifteen packets. Due to the increased number of data samples, the accuracy of the bottleneck bandwidth estimation is increased over standard packet-pair probing (which uses only two packets).

Finally, the Packet-Pair technique is applied to dynamic server selection by Carter and Crovella [12]. WWW document data is duplicated across a given network and when a request is made, the least congested server/connection is selected for delivery. This selection is made using a lightweight Packet-Pair probe, *bprobe*, which sends at most 1% of the requested document size in probe packets. This limit is imposed in order to minimise the congestion impact of packet probing in relation to real network data. Their experiments prove the functionality of this method, and highlight the benefits of dynamic HTTP server selection.

2.3 High Speed Networks

The range of physical and network layer technologies upon which a connection may run, has expanded greatly over recent years. This has prompted work in a number of areas, but particularly the use of alternative congestion avoidance algorithms. The need for these has been highlighted by technologies such as ATM (http://www.atmforum.org) and Satellite connectivity. When using such technologies, it is important to consider their unique properties. In ATM, for example, it is possible that congestion avoidance algorithms and buffer allocations may conflict with the efforts of the transport layer TCP [13]. Furthermore, a wide variety of network layers, means that TCP has to operate with varying degrees of packet fragmentation.

ATM operates on fixed length 53 byte cells (48 bytes of data, 5 bytes of header), which has been noted to cause problems for standard TCP implementations [6]. In particular, the loss of a single ATM cell will incur the retransmission of an entire TCP segment, when running TCP over IP over ATM. Due to the fragmentation of IP incurred when running over ATM, a single IP packet may consist of tens of ATM cells. Similarly, a TCP segment may consist of several IP packets. Should a single ATM cell be lost, then an IP packet will become corrupt. This corruption also reaches the TCP layer, where an entire TCP segment will not match its checksum upon receipt at the remote host. The receiver will then be unable to acknowledge receipt of the data and it will have to be retransmitted.

Further problems arise with ATM when congestion occurs and cells are dropped. Should an individual cell be dropped, the implication for IP packets or TCP segments is much greater. With part of its data having been lost, the remainder of the packet in flight is essentially useless. Therefore, a great deal of work has been carried out to address the issue of packet drop policies in ATM services. Among those most commonly implemented in ATM hardware are Early and Partial Packet Discard (EPD/PPD), as recommended in [43]. In PPD, an ATM switch will drop the remained of a packet should any of its constituent cells be lost. Conversely, EPD suggests that entire packets should be dropped before congestion reaches a critical level. In their experiments, it was shown that EPD gave higher overall performance due to its bandwidth saving techniques.

However, it remains for the higher level transport protocol to recover from any cell loss at the ATM layer. A further problem arises due to many ATM connections being given a guaranteed QoS connection. This implies that a customer may be paying a premium fee for their network facilities. Therefore, it is important to maximise usage of the available bandwidth.

With QoS in operation, a given connection may have pre-determined, allocated bandwidth and guaranteed jitter/delay bounds. Therefore, any packet loss should certainly be transient in nature and fall within the specification of the customer's service level agreement. Having a TCP, therefore, which performs a lengthy timeout and slow start under such circumstances, is not desirable. Provided the cell loss was caused by transient network congestion and not hardware or network management issues, the TCP connection should resume transmission at the optimum rate as soon as possible.

Mechanisms such as Fast Retransmit and Recovery will certainly help TCP in the above situation, as they minimise the number of timeouts and help TCP to sustain the optimum transmission rate. However, should timeout occur, further adjustments may be required in order to maintain the flow of data. In our work with TCP congestion control modifications, [49], this issue is addressed with modifications to TCP's congestion avoidance algorithms. A given TCP connection maintains an *average* congestion window value, which is used should it timeout and have to restart. TCP is then able to restore its connection at the rate prior to any packet loss without having to go through slow start. Should congestion persist, the TCP modification monitors the number of ACK's received after connection timeout and will drop back to standard slow start if none are received within a given period.

2.4 Summary

In this chapter, we have surveyed work related to the evolution of TCP and alternative transport layer protocols. Our research showed that a large percentage of transport protocol development was taking place in producing alternatives to TCP, "TCP Variants". Work published by Brakmo et. al [11] introduced TCP Vegas and pro-active congestion avoidance algorithms to the transport layer. The results presented in the aforementioned paper show that pro-active algorithms are capable of significantly reducing the packet loss that a data stream will experience on congested networks.

Work conducted by Keshav [26] into packet-pair probing, allows a protocol to calculate the current network bottleneck capacity on a given path. This value can then be used to initiate transfer between two network hosts, at a rate which is conducive to current network traffic.

At the application layer, increased deployment and utilisation of bursty, Web-based applications, has produced a requirement for application-focused Quality of Service (QoS) [9]. A side effect of deploying QoS is that a given data stream may have end-to-end bandwidth reservation and delay guarantees from intervening network switches and routers. At the very least, it will be assigned a queueing priority.

This leads us to a coherent research programme that investigates the potential for QoS support at the transport layer. Furthermore, we aim to provide a transport layer protocol which exhibits lower packet loss than current TCP implementations, while maintaining packet goodput [1]. Section 3.2 in chapter 3 discusses the rationale behind and benefits of this research.

Chapter 3 describes our prototype transport protocol and the environment for our simulation testing. The results of our simulations along with traffic and topology specifications are subsequently presented in chapter 4.

[1] Packet *goodput* is the rate of *successfully received* packets at a remote host. This is opposed to *throughput*, which is simply the rate of packet transmission from a TCP sender.

Chapter 3

STTP: Rationale and Design

In chapter 2 we outline a programme of research which addresses deficiencies in current transport protocol (TCP) implementations. The aim of our research is to improve performance while providing support for network Quality of Service. In order to address the bursty nature of Web (HTTP) and multimedia data streams, the experiments carried out in chapter 4 use real traffic traces as input to transmitting sources.

This chapter discusses the design and implementation of our experimental protocol, STTP. The design work detailed here addressed contribution C2, as outlined in chapter 1. Our work with simulated and prototype network protocols will be described (section 3.1) in addition to giving details of the simulation packages used (section 3.3). The rationale for our research is explained in section 3.2. We then outline the framework and implementation details of STTP (section 3.5) and provide selected key performance results. Additional experiments, graphs and data can be found in the appendices. The information presented here has been published in [49], [51] and [50], which will provide the reader with further background detail of our work.

3.1 TCP Modifications

Initial work for key objective areas K1 and K2 was focused on the improvement of TCP Reno, the most common implementation of TCP at the time of writing. This work is outlined by contribution C1 in chapter 1. High bandwidth and Quality of Service-aware networks can present the user with a reliable, sustainable allocation of bandwidth. The physical layer for such connections is invariably fibre optic cable for the majority of land-based communication, or satellite/microwave channel for mobile or air-based networks. In both types of network, packet loss is bursty and not generally sustained due to prolonged congestion.

Standard TCP Reno will throttle its transmission rate and perform a slow start when packet loss occurs or multiple duplicate acknowledgement packets are received. In a high bandwidth environment, this can be damaging to the communication stream between two hosts due to the time it will take TCP to resume transmission at the rate prior to packet loss.

Therefore, in [49], we proposed modifications to TCP Reno congestion avoidance algorithms which addressed these issues. A feature was implemented to maintain a history-weighted average of the sender's congestion window size. When the congestion window was updated, the average value would be re-calculated with, for example, $(0.1 * new\_window\_size + 0.9 * current\_average\_window\_value)$. The effect of this weighting was to reduce rapid fluctuation in the average window size when bandwidth became suddenly available or reduced.

The average window value was then used as the restart value when TCP Reno encountered packet loss. The protocol was then able to resume transmission at the average window rate it had achieved in the history of its current conversation. In order to prevent further packet loss in case of sustained congestion, the algorithm was engineered to fall back to traditional congestion avoidance techniques should packets not be acknowledged after a restart had taken place.

These modifications were simulated in the REAL simulation package (see subsection 3.3.1). Results showed a performance increase of up to 46% in highly congested conditions, using lengthy, sustained, transfers of FTP data. Tabular and graphed and simulation data is available in [49]. In this report, we propose extensions to TCP congestion control which, on a congested network result in significantly better use of available bandwidth by eliminating the requirement for a *slow start* with each TCP restart.

Mechanisms are implemented which enable the sending TCP to restart its data flow at a suitable level for the current connection. A *fallback* mode is provided to prevent the source from overloading intervening routers should congestion be sufficiently high.

Rigorous testing of the new algorithms was undertaken using the REAL network simulator and various benchmark scenarios. In addition to the *benchmark* scenarios, further models were developed in order to simulate *real-world* situations.

Over the suite of tests, our modifications showed on average a 20-30% speed increase over REAL's standard TCP-RENO protocol (which is based on BSD's TCP-RENO) with some sources showing up to a 100% improvement. In the worst-case scenarios, the modified TCP functioned at least as well as TCP-RENO.

3.2 Rationale for Research

In recent years, a great deal of work has been conducted, aimed at addressing the issues posed by the transport of multimedia data with an increasingly complex protocol stack. Prior applications incorporating bulk data transfer often involved a simple connection setup followed by lengthy (whole seconds or minutes) of TCP over IP communication. The World Wide Web for example, is an object-oriented environment, in which the user is normally required to download several components in order to view a single page item (HTTP object). With the current HTTP specification, this results in several short downloads, each requiring a separate TCP connection. On a high-bandwidth connection, the download time for each component may be below one second including setup and tear down.

In [49], we examined the most common existing TCP implementation, TCP

Reno, and discussed ways in which it could be improved to accommodate high speed and networks with support for Quality of Service. However, the framework itself, with exponential slow start and lossy, reactive congestion avoidance, could not be improved without substantial redesign.

Therefore, our studies led to further practical examination of this area. We believe that there exist a number of improvements which can be made to the general profile of a transport layer connection. Namely, the startup phase and the protocol's reaction to network congestion. This work is outlined in area K3 of our thesis objectives in chapter 1.

Our work in both flow control and network bandwidth probing [50] [51] showed us that more suitable mechanisms were available for bursty, multimedia traffic. The token bucket model allows bounded burstiness, but with a mean transmission rate being enforced over a given time period. Data may be transmitted from an STTP source provided that there is credit in the token bucket. If not, transmission will commence when sufficient credit has been accumulated. Credit is added to the bucket in regular "drips" from a timed source.

Provided such a model can be initialised with values which will not overload a network path, or hinder data transmission, it would appear to be well suited for flow control in high speed networks with bursty traffic profiles. Furthermore, the relatively simple parameters of the token bucket model mean that the rate at which transmission occurs can be adjusted in real time to take into account variation in network congestion. The bucket's burst size is an integer value, representing the number of tokens it is capable of holding, and the mean flow rate is a timer value for the introduction of new packets to the holding area.

In order to initialise the bucket with appropriate values, we draw on Keshav's work with packet-pair bandwidth probing [26]. This allows us to initialise the token bucket parameter with suitable values in roughly a single round trip time.

Once established, a modified pro-active congestion avoidance algorithm is used to adjust the flow rate and burst size of the token bucket. Using measured packet round trip times, we can sense variation in queuing delay and make proportional adjustment to a connection's flow parameters.

The result is a flexible transport protocol which is capable of quickly and intelligently measuring the available bandwidth on a network path. It can instantly proceed to transmit bursty data which is bounded by the current network capacity. However, over a given period, it will exhibit fairness with a mean flow comparable with that of competing connections. We demonstrate this functionality in chapter 4, section 4.2.2, where STTP, TCP Reno and TCP Vegas are simulated in the REAL network simulator.

3.3 Network Simulators

In order to experiment with alternative algorithms under a wide variety of scenarios, we elected to use network simulation software. Our requirements for such software were relatively demanding since we needed full control over and access to the simulator core. Therefore, our options were limited to the freely available packages from other research groups or networking projects.

Network simulation software was chosen for its ability to construct a wide range of topologies with a variable number of communicating hosts and routers. Network hosts can then be configured to transmit a range of traffic types, from FTP, HTTP, Telnet through to mathematically distributed data profiles, such as the poisson and exponential models. Similarly, most current simulators will allow the user to provide their own traffic traces which can then be used by a network host to provide input for its data transmission.

The ability to construct such networks is of great value when examining or designing network protocols. If only a single LAN topology were available, then the user could not be certain of the protocol's performance on wide area or low bandwidth networks, or under high congestion.

A network simulator such as NS [5] or REAL [25] can provide these facilities, but is only as reliable as its internal models and program code accuracy. It is also difficult for designers to model the random traffic patterns and true burstiness of an

Internet. Published protocol work at this time relies mainly on mathematical and simulation models. Not everyone chooses to progress through to the implementation and prototyping phase. The results from our simulation work with STTP, shown in chapter 4, demonstrate how effective and consistent network simulation has become.

3.3.1 REAL

When work began on this thesis in 1996, there were few packages stable and developed enough to enable rigorous testing with reliable protocol suites. At this time, the REAL simulator version 4.5 [25] was widely used, well documented, and had been the foundation for significant published work [26], [27], [28], [30], [31], [29]. REAL is based on the NEST 2.5 (ftp://ftp.cs.columbia.edu/nest/) simulation testbed and was used as the basis of the NS simulator created by Floyd and others at the University of Berkeley [5].

The simulator itself is written entirely in C, but the user describes network topologies using a simple description and scripting language. This is a simple, yet efficient approach which has been employed by many simulation packages in the past. Version 4.5 of REAL was used by many research groups for simulation experiments. However, Version 5.0 released in 1997, introduced several new features and offered significant speed improvements. The majority of our work was conducted using version 4.5 of the software and while it was relatively simple to port development code between releases, we continued to do so after 5.0 was available. Despite 5.0 offering more features, extensive use of the new code showed 4.5 to be both more stable and less prone to erratic behaviour. Our results from version 4.5 were confirmed by further experimentation in NS.

STTP was implemented in C as part of the REAL core simulator. Once compiled, we were able to experiment with it alongside TCP Reno using user-side scripted scenarios.

3.3.2 NS

In contrast with NS, REAL does not allow further functionality to be included in its user-side scripts. These are used simply to describe the network topology and events which will take place during the course of the simulation.

NS began as a variant of the REAL simulator in 1989, but did not see significant development effort until the late 1990's. Version 2.0 was released on September 10th 1997. The current release at the time of writing (1999), 2.1b5, was used for our simulation experiments. NS development is progressing, and further releases should now be available to the reader.

NS is an event-driven simulator, which consists of C++ core methods, which interface with an object-oriented Tcl (oTcl) shell. This powerful framework allows the user to implement both network topologies and additional functionality through methods and procedures in oTcl scripts. For example, in order to monitor the behaviour of specific variables within a protocol, the user has only to present these via the oTcl interface in the protocol's core C++ code. An oTcl simulation script is then able to read and manipulate these variables during runtime.

The implication of this is that experimental prototypes can easily be created (using oTcl rather than a full C++ implementation), by calling on features or methods already implemented in the simulator core. However in order to create or modify fully functional network protocols, it is better to build them in C++. The reason for this is that while an oTcl prototype may be quick, it is not as integrated nor can it offer the same level of functionality as a full implementation.

Variables and methods are made available to user-side scripts through the oTcl interface, which consists of C++ function calls from the oTcl libraries. It is therefore possible to allow or deny user access to protocol data as required. While this does facilitate program debugging and monitoring, the interface is relatively complex and not as intuitive as that of REAL, for example. Due to the inclusion of oTcl, both on the user-side and as function calls in the C++ code, the learning curve for NS is quite steep. Once overcome, however, it provides a powerful and flexible simulation environment. The NS simulation model of STTP is implemented entirely in C++, but presents many variables to the oTcl interface. This permitted us to trace important variables such as the Congestion Window size during the simulation.

3.3.3 Simulator Validation

Both REAL and NS are provided with a large collection of validation and benchmarking user scripts. NS, in particular, performs self validation as part of the build process.

A detailed set of scripts are provided, which test application, transport, routing and link-layer protocols. The simulator is run using known input values and the output compared with known, valid results. The validated set of protocols extends to all common Internet standards, with a smaller set of non-validated, yet working code. As new builds of NS and its protocols are produced, the package is validated by its maintainers on a variety of platforms. The user is notified should NS fail to validate any of its protocols during the build process. This process was used to validate the functionality of TCP Reno and TCP Vegas for the purposes of our simulation experiments.

Keshav modelled and described REAL in great detail, in [25]. For many years, REAL was one of the premier network simulators in the academic community, only now superceded by NS (which is derived from REAL).

Both simulators have been used extensively by other research groups and have formed the basis for a great deal of published work. The REAL simulator was designed and validated as part of Keshav's thesis [27]. NS is currently being used by many PhD research students and networking groups. Full documentation and example program code for NS can be found at http://www.isi.edu/nsnam/ns/. Published background information on NS can be found in [5].

3.4 Framework/Algorithms

STTP itself is a reliable transport layer, intended for use on IP networks in place of TCP. The algorithms, however, are portable and could be used over any data-link layer.

The acronym, STTP, stands for Shaped Token Transport Protocol. This is taken from the design of its core algorithms, which are centered around the use of token and leaky buckets, used to shape the flow of data from a transmitting network host. By initialising these buckets with appropriate values and maintaining them in accordance with the current network state, we are able to provide a shaped flow of data. It is anticipated that this flow will be more conducive to the support of Quality of Service (QoS) network applications.

We therefore have three main areas of concern, detailed below in paragraphs 3a-3c:

3a An STTP connection is initialised with a dual-packet probe from the transmitting host to the receiver. This will inform the host of an incoming connection and allow it to prepare input buffers and start application server functions. The probe consists of two packets, transmitted back-to-back, which are used to measure the bottleneck bandwidth on a given network path. This is described in detail in subsection 3.4.1.

Provided that the Maximum Transfer Unit (MTU) of the path is known, we are now able to calculate the speed at which new packets can be put onto the link in order to fill the available bottleneck. The result of this calculation is used to trigger the release of tokens into a token bucket. The functionality of this flow control mechanism is described in subsection 3.4.2.

3b The token bucket mechanism is used as a flexible replacement for TCP's Congestion Window. This addresses key area K3 of our objectives. We are able to modify the flow of tokens into the bucket in real time as we receive network congestion information back from acknowledged packets. In order to achieve this, we use a pro-active mechanism as described in subsection 3.4.3.

Our algorithms monitor the round trip time (RTT) of transmitted data, and respond to increases or decreases in network latency. STTP is an acknowledged protocol, which means that each packet received at a remote host will generate a small acknowledgement packets. This is returned to the transmitting host as proof of receipt. These packets are generated and returned immediately by the remote host.

If a connection's RTT falls outside a bounded window when compared with previous measurements, then the token bucket's timer is modified accordingly. For example, if the RTT should increase, then this can be interpreted as congestion on this connection's network path. We can therefore decrease the rapidity of the token bucket timer in proportion to this change. The result is reduced packet transmission in line with the congestion currently being experienced. Similarly, if the RTT decreases, we can increase the rate of packet transmission.

3c In the case of packet loss, or multiple duplicate acknowledgement packets, STTP adopts the same approach as TCP Vegas, by reducing its transmission rate by 10%. This addresses item K3 of our objectives. However, as can be seen in our experimental results, STTP drops far fewer packets than either TCP Reno or Vegas [51].

3.4.1 Packet-Pair Bandwidth Probing

While TCP Reno's standard slow start algorithm could have been used, past research has shown it to be lossy and poorly suited to multimedia applications [16]. Packet-pair probing techniques have been examined in work by Keshav [26] and others over recent years. While there are some reservations as to its use in certain network configurations, our work has shown it to be reliable in almost all common cases with both traditional and current traffic profiles. It has been reported that packet-pair reliability is reduced when using certain types of router queueing algorithm and network link asymmetry [26]. Keshav describes a scenario where traditional First Come First Server (FCFS) router algorithms present problems to packet-pair network probes. If a single source were to send a large burst of data to an FCFS router, then according to the scheduling algorithm, it would receive a higher priority of service than competing flows. This is due to the transmitting source sending a large number of packets in quick succession, which will be queued and processed in order by the router. In [26] Keshav recommends that Weighted Fair Queueing (WFQ) is more appropriate for use with packet-pair network probe techniques. This algorithm avoids the problems associated with FCFS by preventing a high-bandwidth source from monopolising a router's processing capacity. Keshav reports that packet-pair algorithms can be adversely affected by heavy traffic at a FCFS router due to the inconsistency in service rate, which is caused by high-bandwidth data flows from competing sources. He argues that WFQ provides packet-pair with a more realistic view of the network state due to each flow being assigned a priority within a router's service model. Each flow will therefore receive its fair share of routing resource.

Allman and Paxson examine packet-pair in [3], where the issue of asymmetric network connections is discussed. Additional modelling and theory behind packet-pair probing is presented in detail in [26], [27] and in our own work, [50], [51].

Simply put, one can transmit two packets back-to-back along a given network path. Given that we know the packet size, the amount by which they are separated by queueing delay at the receiver allows us to calculate the current bottleneck link capacity.

The formula used to calculate the bottleneck bandwidth on a given connection is:

$$bottleneckcapacity(bits/second) = PacketSize(bits)/InterarrivalGap(seconds)$$

The procedure for initialising an STTP connection is therefore quite simple. Two packets are transmitted back-to-back, i.e. queued and transmitted as close together as the network adapter driver will allow. When received by the remote host, they are simply echoed back to the receiver without delay. When the packets arrive back at the sender, the gap between their arrival is measured and used in the above calculation.

This method is termed *sender-side bandwidth* probing by Allman in [3]. When using asymmetric connections, or multiple bonded channels, it would be advisable to use receiver-side measurement. An added advantage of this, as discussed by Allman in his work and Keshav in [26], is that the probability of error is halved when using only the inward or outbound path. For the purposes of our simulations, we have used only sender-side bandwidth probing. The reason for this is that our simulation topologies do not incorporate asymmetric links, and while the benefits of receiver-side measurement will certainly yield better results in real networks, our implementation of the packet-pair probe proved to be reliable and accurate.

The bottleneck measurement can then be used to calculate an appropriate burst and feed value for a connection's token bucket. During our experimentation, we used a variety of methods which will be discussed later in this section.

3.4.2 Token Bucket Flow Control

Once a network connection's bandwidth has been discovered, a token bucket can be initialised with an appropriate flow of tokens and initial burst size. For our work, the standard $\sigma - \rho$ model has been used, where σ is the capacity of the bucket (a connection's maximum burst size) and ρ is rate at which tokens are permitted to enter. At any time, an application is only able to send data if there are tokens in the bucket. In our simulations, we used fixed packet sizes and therefore made each token in the bucket equal to a single packet of network data. In a kernel implementation, however, it may be advisable to use an integer value for the bucket and allow variable-size packets to be transmitted. This would accommodate small client requests to a remote server. In order to scale the token bucket in accordance with the current network state, both the flow rate and burst size must be altered in real time.

During our simulation experiments, we used two distinctly different approaches to token bucket management. The *first*, and initial implementation, did not rely on traditional timers to trigger the release of tokens. Instead, we used a mechanism by which the acknowledgements received at the sender were added to a leaky bucket. The leaky bucket was then responsible for feeding tokens to the token bucket at the current bottleneck rate. Our leaky bucket implementation incremented its current value with each new ACK packet received. With the advent of a timer event, the leaky bucket is decremented and the token bucket incremented to indicate the transfer of a token.

The problem with this approach is that it requires the token bucket to be initialised with an appropriate number of tokens. These are needed to *kick start* the STTP connection by providing a number of tokens to the newly opened connection. When in operation, the number of tokens in operation can be increased or decreased according to the current state of network congestion. A further variable for tuning is the flow rate of the leaky bucket component as this is responsible for the smoothing of token flow.

The advantage with this mechanism is that tokens will only be fed into the token bucket if packets are being successfully received and ACKs generated. Should there be a sudden influx of network congestion and ACK packets do not arrive at the transmitting host, then STTP will not blindly inject packets into an already congested network.

The *second* approach used was that of a traditional token bucket, which injected a token into the bucket with each timer event. The timer was initialised to the rate required to fill the probed bottleneck connection. For example, a 64kbps bottleneck connection would yield a timer that generates 64000/packetSize(bits) events each second. Each time an event occurs, credit for one packet transmission is added to the token bucket.

The latter was found to be the more elegant and appropriate solution. However, the difference in overall performance between the two in our simulations, was found to be negligible. Due to the random nature of real network traffic, we believe that a kernel implementation, and testing on a physical network would provide more detailed data. The second approach was a more accurate implementation of our STTP design, and so was used for the experiments in chapter 4.

3.4.3 Congestion Avoidance

Research has shown that pro-active congestion avoidance is both fair and less prone to packet loss than traditional TCP Reno algorithms. Early work by Wang and Crowcroft [52], demonstrated the benefits of this approach. However, the most significant work in this area was conducted by Brakmo et. al in their implementation of TCP Vegas [11]. Their results showed significant performance improvements over TCP Reno when considering overall throughput and packet loss.

TCP Vegas made approximations as to the anticipated throughput that would be achieved in a given time period (one Round Trip Time). If the actual throughput in this period was lower than expected, then the protocol took this as an indication of network congestion. It therefore reduced its transmission rate accordingly. Conversely, if actual throughput was higher than expected, then the transmission rate is increased.

This approach to congestion avoidance also lends itself to token bucket flow control. Rather than modelling the expected and real throughput values, we chose to monitor the first order statistic of connection Round Trip Time (RTT).

When an STTP packet arrives at the receiver, an ACK packet is generated and returned to the sender. A single RTT is the time it takes for the data to arrive at its destination plus the time for the relevant ACK to reach the sender.

A clear indication of network congestion, or outage, is an increase in RTT. This is due to additional packets being queued at routers along a given connection's path. Should router queues overflow, then packets will have to be dropped as they cannot be accommodated in router memory. In contrast, TCP Reno does not sense network congestion, and continues to send at its present rate until data is lost. It then reacts by reducing its transmission rate.

In order to prevent rapid fluctuation in a connection's transmission rate, we provide bounds to STTP's RTT monitor. For experimental purposes, we chose 5%, as this mechanism is also used by TCP Vegas. Therefore, if STTP sees a connection's RTT change by greater than 5% compared with the last monitored value, it will reduce both σ and ρ by an amount proportional to the change in RTT. Experiments were conducted with values between 1% and 20%. However, with small values, the protocol became too sensitive to variation in RTT, and vice versa with large values (greater than 10%). We found 5% to give the best combination of sensitivity and stability for our experiments. The following function, called each time a new ACK is received, describes the modification of token bucket depth (σ) and flow rate (ρ).

```
lastRTT is initialised to 0.0
newRTT is set to the current measured RTT
IF (lastRTT > 0.0)
    IF (newRTT > lastRTT*1.05)
        decrease depth and flow of bucket
    ELSE IF (newRTT < lastRTT*0.95)
        increase depth and flow of bucket
lastRTT := newRTT
```

This reaction to RTT variation means that the flow of data from a STTP source is scaled in proportion to the available bandwidth on a given connection. We will discuss the relative advantages and disadvantages of this approach in chapter 5. However, further to our work in [49], which introduced the notion of smoothed window calculations, this technique is used in the real time manipulation of STTP data flow, as this technique is outlined in section 3.1 of this chapter.

3.4.4 History Weighted Bucket Manipulation

In [49], we developed a mechanism by which a smoothed, average value for TCP's congestion window could be maintained. The specifics of this technique were discussed in chapter 3, section 3.2. Our simulation experiments showed that this technique was required in order to prevent rapid fluctuation of window size in congested networks. By maintaining a history-weighted congestion window value, the effect of bursty network congestion can be minimised, as TCP can attempt to restart at its mean rate rather than with a single segment slow start upon packet loss or timeout.

Our experiments with STTP in both REAL and NS, showed that this approach to transmission rate management was also applicable to token bucket flow control. Should the RTT of a connection change by ±5%, then a recalculation is required. In a similar vein to our TCP modifications, if high-bandwidth, bursty, cross traffic is being experienced, then it may be short-lived, as most HTTP or multimedia transfers are. Therefore when calculations are performed, the current token bucket values are given a higher value, or weight, than those newly which have been measured. The method we have used is to adopt two variables, α and β, where $\alpha+\beta = 1.0$. α is used for the history weighting, and β for the newly measured value. Initial experiments used a simple increase or decrease calculation which was proportional to the change in measured RTT.

$$\mathcal{F} = 1 + lastRTT/newRTT \quad \sigma = \sigma\mathcal{F} \ \rho = \rho\mathcal{F}$$

A history weighted calculation includes terms where α and β are used to modify the relative importance of these measurements.

$$\mathcal{F} = \alpha + \beta \quad \sigma = \sigma\mathcal{F} \ \rho = \rho\mathcal{F}$$

The effect of using this technique is to reduce fluctuation in token bucket values. We found that high-bandwidth cross traffic caused a sudden increase in the measured RTT of STTP connections. This resulted in a rapid reduction in a given connection's transmission rate. Conversely, when competing sources on a network completed transmission, the measured RTT would suddenly reduce. STTP would sense this and increase its traffic flow accordingly. Particularly in cases where on-off or bursty traffic sources were in use, this is undesirable behaviour.

Rapidly fluctuating transmission rates are not network-friendly, nor desirable from a user perspective with regard to application QoS. STTP therefore smooths its transmission rate adjustments using the above technique. The details of our work with this technique are published in [51], but summary results are included in chapter 4, section 4.2.2.

3.5 Protocol Implementation

While the TCP modifications described in section 3.1 [49] were implemented in the REAL simulation package, the later stages of our research were performed using NS. Due to the large amount of development currently taking place for NS, we found that its support of up-to-date network protocols and technologies was far superior to that of other simulation packages. In order to test STTP with a broad range of traffic types, network configurations and TCP implementations, NS was an obvious choice.

A simulation prototype of STTP was implemented in the NS network simulator. The purpose of this was to implement our protocol model in a familiar simulation environment. We were then able to test its functionality and show that further work would be valuable given initial performance results. The work done in [51] used NS as the algorithms and experience gained in REAL were ported to the new simulation package.

In both cases, standard TCP Reno was used as the basis of our implementation. By adopting the basic algorithms for packet processing, transmission and retransmission, we were able to more faithfully test our adjustments and improvements.

TCP has three basic sections to its program code: *Packet transmission*, packet receipt and timeout processing. Packet transmission ensures that packets are sent in sequence order and at a rate which is in line with the bounds described by a connection's congestion window variable. *Packet receipt* processes incoming ACK's, and is responsible for detecting out-of-order or duplicate packets. When an in-order ACK is received, the appropriate adjustment to a connection's congestion window size is made. This function will, however, note duplicate ACK's to a point where fast retransmit is triggered, or a connection is restarted. When transmitted, TCP segments have a *timeout value* assigned. Should this expire before the appropriate ACK is received, telling of successful delivery, then it is retransmitted and normally accompanied by a slow start due to supposed network congestion.

In order to facilitate our implementation, the retransmission and timeout code from TCP Reno was incorporated into the STTP framework. This provided an even basis for comparison when considering retransmitted or lost data. If any improvements were apparent, it was therefore due to our flow control, congestion avoidance or startup algorithms.

3.6 Summary

In this chapter we have addressed the design aspect of K3, as detailed in chapter 1. We have also covered the material related to C1 (section 3.2) and C2 (throughout this chapter). The simulation of STTP (K3 and K4, resulting in C3), is presented in chapter 4. The results of this experimentation is discussed in chapter 5 (K4 and C4).

Chapter 4

STTP: Testing and Results

In this chapter, we present results from network simulations using NS and REAL, with an implementation of our protocol model, as described in chapter 3. This work is outlined as thesis contribution C3 in chapter 1. We will proceed with detailed discussion of the graphical and tabular results presented here, in chapter 5.

This chapter comprises the functional and subsequent performance testing of our experimental protocol, STTP. Two simulation packages are used, REAL and NS, which provide a wide range of comparative protocols and application traffic types.

STTP's algorithms were implemented and built as part of both simulation packages. This allowed experimentation alongside other common variants of TCP, Reno and Vegas. The current de facto standard for TCP/IP networked systems is TCP Reno. However, a great deal of work has been conducted into the research, development and study of alternative congestion avoidance algorithms such as TCP Vegas. The resulting techniques are becoming increasingly common in a large number of Operating Systems such as Linux (http://www.linux.org).

The REAL simulator does not have an implementation of TCP Vegas as part of its standard distribution. We therefore performed functional testing of STTP against TCP Reno with bulk data flows. Further work with NS allowed more detailed performance testing with TCP Reno, TCP Vegas and STTP, using different traffic types. Tables 4.2 and 4.1 show a summary of the features of both NS and REAL in terms of protocol and traffic type support. More detailed information can be found from the project development homepages (http://www.cs.cornell.edu/skeshav/real/ for REAL, and http://www-mash.cs.berkeley.edu/ns/ for NS).

Table 4.1: Summary of REAL and NS Simulator Traffic Types

Simulator	TCP Tahoe	TCP Reno	TCP Vegas	Packet Pair
REAL	x	x	-	x
NS	x	x	x	-

Simulator	FTP	Telnet	Statistical	User Traces
REAL	x	x	x	MPEG only
NS	x	x	x	x

Table 4.2: Summary of REAL and NS Simulator Protocol Support

Both simulation packages allow the user to obtain statistical information regarding the number of packets transmitted and dropped by each traffic source. Given these values, it is possible to calculate the *goodput* for a given connection. The term goodput is used to describe the rate at which packets have been successfully received at their destination. With a guaranteed delivery mechanism such as TCP, the higher the goodput, the more efficient the transport layer. A transport protocol which is reactive to network congestion and conservative with packet transmission, is likely to have a higher level of goodput than one which is aggressive and eager to capture available bandwidth. This is shown in experiment 4.2.2, where a range of values are simulated in NS with our STTP model.

The majority of our more advanced simulation scenarios were implemented in the NS simulation package, due to the flexibility that it offers in terms of user traffic types and scripting functionality. Our work focused on the implementation and improvement of congestion avoidance algorithms at the transport layer, and in order to rigorously test our models, custom scripts were developed during the course of our simulations. In particular, we had the need to run batches of simulations with varying degrees of network congestion. This was achieved by running the same simulation with an increasing number of transmitting and receiving nodes, which were automatically added to the simulation script files between runs. Each simulation would produce an output file, which could later be parsed with a simple Perl script, and input to Gnuplot for presentation.

4.1 Testing STTP Network Behaviour

In this section, simulation experiments are conducted with the REAL simulation package. The aim of this work is to validate the protocol model and to compare initial performance with that of existing TCP implementations using simple data transfer.

Experiment #	Demonstrates
1 (section 4.1.1)	The functionality of STTP's congestion avoidance algorithms and flow control
2 (section 4.2.1)	The performance of STTP with bulk data transfers
3 (section 4.2.2)	The fairness of STTP in comparison with TCP Reno and TCP Vegas
4 (section 4.2.3)	STTP and TCP performance with Variable Bit-Rate video sources
5 (section 4.2.4)	STTP and TCP performance with bursty HTTP traffic sources
6 (section 4.2.5)	The performance of STTP, TCP Reno and TCP Vegas in a mixed protocol environment

Table 4.3: Experiment Information

4.1.1 Experiment 1, Functional Testing using REAL

Figure 4.1 depicts a network topology with several sources (transmitting nodes) and a single sink (destination node). The traffic traverses two routers with all links being 64kb/s capacity.

In order to examine STTP's bandwidth probe and congestion avoidance algorithms, six STTP sources were started at regular intervals (T=0,10,20..50). The implication being that subsequent sources would be probing into a busy connection and so have to compete with other sources for available bandwidth. Similarly,

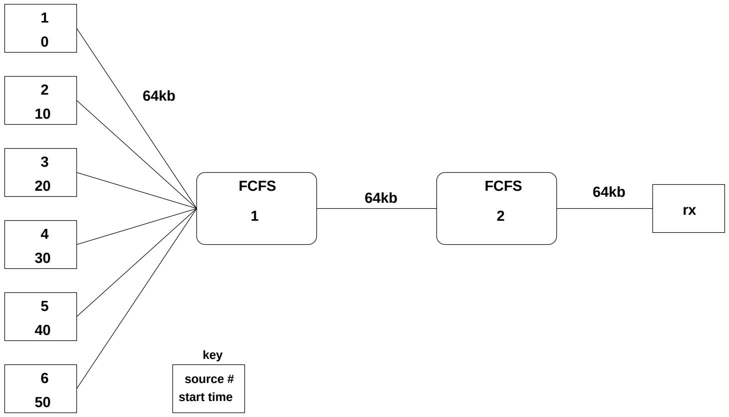

Figure 4.1: Experiment 1

existing traffic would have to relinquish bandwidth in order to accommodate new sources. Furthermore, each end node has a dedicated 64kb/s connection but which is of the same capacity as the network's shared segments. This implies that any bandwidth probe will be required to probe bottleneck segments in the main network and will always encounter links which have a greater capacity.

node	transmits(pkts) [sd]		drops(pkts) [sd]		RTT(ms) [sd]	
	Reno	STTP	Reno	STTP	Reno	STTP
1	34.56 [19.52]	37.89 [17.55]	24.89 [38.17]	0	1.89 [3.00]	3.71 [1.9]
2	39.00 [15.99]	37.56 [17.08]	14.11 [23.05]	0	2.13 [3.43]	3.71 [1.9]
3	28.33 [20.27]	13.13 [2.48]	3.89 [8.54]	0	2.67 [2.81]	2.66 [1.19]
4	23.89 [15.52]	12.77 [2.95]	11.44 [17.15]	0	4.09 [4.95]	3.12 [1.95]
5	21.67 [16.31]	13.07 [2.97]	5.22 [11.11]	0	3.61 [4.73]	3.11 [2.49]
6	12.56 [17.30]	13.22 [1.63]	1.44 [3.74]	0	1.05 [2.23]	1.75 [1.65]

Table 4.4: TCP and STTP Performance Summary

Table 4.4 shows raw transmission, drop, retransmission and RTT data for both STTP and TCP Reno run on the same network topology and simulation model. In all columns, [*sd*] indicates the standard deviation encountered across data samples. While TCP transmits a similar number of packets to STTP, the actual *goodput* of STTP is significantly higher due to there being no dropped or retransmitted packets. Particularly sources 3, 4 and 5 in Appendix table 4.4 exhibit greatly improved performance, even when an error margin of ±5% is taken into account on our simulation results. It is important to note that the values given in these tables are the average over the duration of an experiment as the complete results are too verbose to include in these pages. The methodology for each of our experiments was to dump all available data to a single file. It was then possible to write Perl or Shell scripts, which would extract the information required for the tables and graph presented here. This proved itself to be a sound methodology, as we did not have to re-run simulations to obtain further information, and all inter-related data was present in a single file. While STTP shows few or no packet drops and retransmissions, TCP has a higher transmission rate. This means that while TCP may transmit more packets, this over-subscription by several sources incurs packet loss and results in a lower overall throughput (goodput) after dropped data is taken into account.

STTP's improvement is due to accurate bandwidth discovery algorithms and proactive congestion avoidance. Our experiments show that STTP is able to initiate transfer at a speed which is appropriate to the current state of network congestion. Furthermore, the results of every experiment show that STTP exhibits lower packet loss than TCP Reno on the same simulation topology. As the RTT of a source increases, STTP will back off its transmission rate so as to avoid buffer congestion at the routers. These results are supported by work done by the NRG at Arizona, when developing the TCP Vegas proactive congestion avoidance algorithms [11], [10].

Figure 4.2 highlights a key feature of STTP. While the summary tables show average, per-flow statistics, we are able to see how each stream performs over the duration of the simulation. Each graph in figure 4.2 follows the same format. With sources starting incrementally, each bar represents the performance of a TCP or STTP connection for a given time period. The x axis is segmented into intervals for the experiment at which data was written to the output file. We can therefore see how each source performs as the simulation progresses and how new connections affect existing traffic.

It is interesting to note in table 4.4 that TCP has (in general) a much higher standard deviation than STTP in its transmission rate. This is highlighted in figure

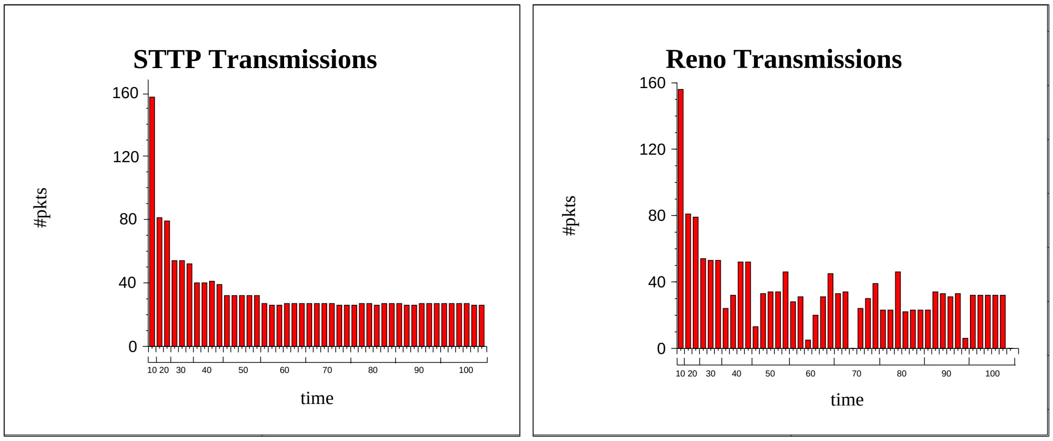

Figure 4.2: Experiment 1: STTP and TCP Packet Transmissions

4.2, where we can see large fluctuations in TCP's transmission rate, even when in steady-state congestion avoidance. STTP, on the other hand, exhibits much smaller variations in its flows and may therefore be considered a *smoother* traffic source.

Considering data in table 4.4, the result of this smoother traffic profile means that STTP has fewer packet drops and could be said to be more *router-friendly*. By reacting before packets are dropped, STTP does not try to send more packets than a link is capable of holding. Conversely, if it detects a decrease in RTT, STTP will increase its packet transmission rate and take advantage of spare bandwidth.

4.2 Performance Testing STTP

Subsequent experiments were performed using the NS simulation package. We chose to move our work to this package due to its expanded functionality and support for modern network protocols. TCP Vegas is included as part of NS's transport protocol library, in addition to rich support for user-defined data streams and trace files. In this section, we test STTP with a variety of different traffic types, including variable bit-rate and HTTP request traffic traces.

4.2.1 Experiment 2, Bulk Data Transfer using NS

The network topology used for the performance testing of STTP can be seen in figure 4.3. Here, we have 1..n transmitting nodes and 1..n receiving nodes. In each case, node t1 transmits to node r1, node t2 to node r2 etc. Similar topologies are recommended by Keshav in his benchmark suite for the REAL [25] simulator, as they permit the rigorous testing of simulated protocols through a combination of congestion paths and dedicated connections. The topologies in themselves are scalable to support many hundreds or thousands of sending and transmitting nodes, and combine the ability to implement bottleneck links and cross traffic in order to test a protocol's congestion avoidance algorithms.

By varying the number of sources and the link bandwidths, we were able to create a wide variety of scenarios under which to test both STTP and TCP Reno. Our comparisons focus on TCP Reno and TCP Vegas, as a large amount of modeling and simulation conducted in the past, provides a clear understanding of their positive and negative attributes.

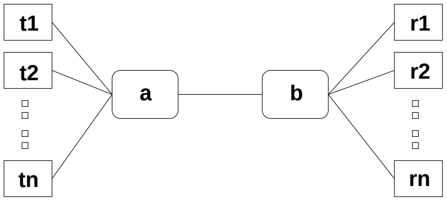

Figure 4.3: Performance Simulation Topology

Experiment 2, the raw data for which is depicted in table 1 uses the same network topology as in previous simulations and bulk transfer, FTP sources. It does, however, cover a large number of sources which is varied from 10 to 200 and for this reason, the backbone link is upgraded from 0.5Mb/s to 2Mb/s. Similarly, to accommodate the additional traffic, the port buffers were increased from 64kB to 128kB. These modifications were required to prevent massive packet loss with a higher number of sources.

Simulations were run with between 10 and 200 sources, incremented by 10 sources

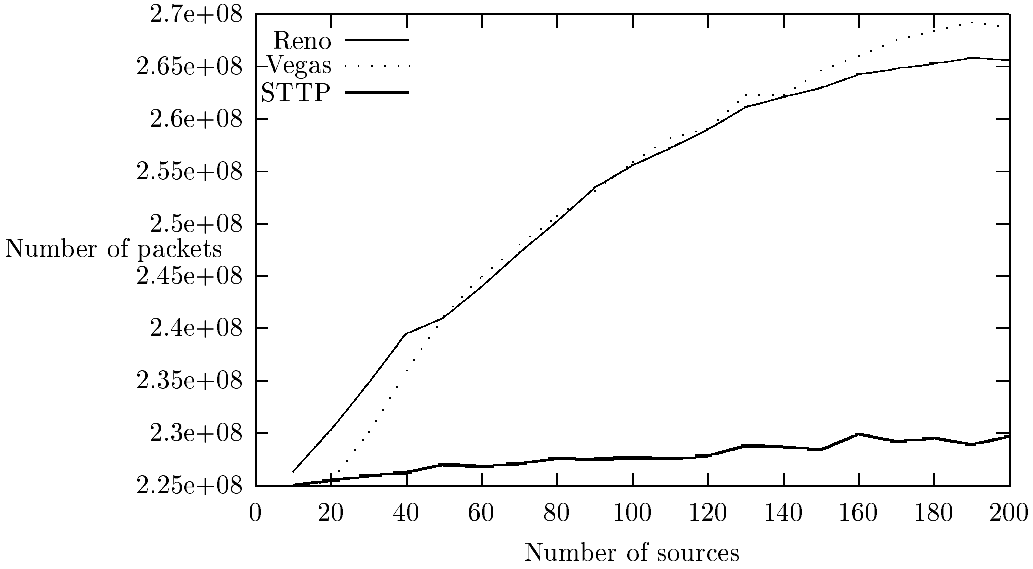

Figure 4.4: Experiment 2: TCP Reno, TCP Vegas and STTP Total Packet Transmissions

with each iteration. Each scenario was run with TCP Reno, TCP Vegas and STTP. The data depicted in table 1 show how many bytes were transmitted, how many were dropped and the mean, high, and low for bytes transmitted. Simulations were run for 900 seconds.

Figures 4.4, 4.5 and 4.6 show the information from table 1 in a graphical format. In each case, the number of packets is plotted on the Y-axis, and the number of traffic sources on the X-axis. Each graph contains the information obtained from the entire suite of experiments, run from 10 to 200 sources for each protocol.

From these graphs, it is clear that while TCP Reno and TCP Vegas have very similar profiles, STTP transmits only 2.25e+08 (10 transmitting sources) to 2.3e+08 (200 transmitting sources) packets (figure 4.4). Compared with the exponential curve, rising to around 2.7e+08, shown by both TCP Reno and Vegas, this shows that STTP is reacting to network congestion and restricting its transmission rate. Figure 4.5 shows that STTP also has much lower packet loss, with between 0 (10 transmitting sources) and 5e+06 (200 transmitting sources) packets dropped, and as a result, has goodput which is comparable to, if not better than, both of the

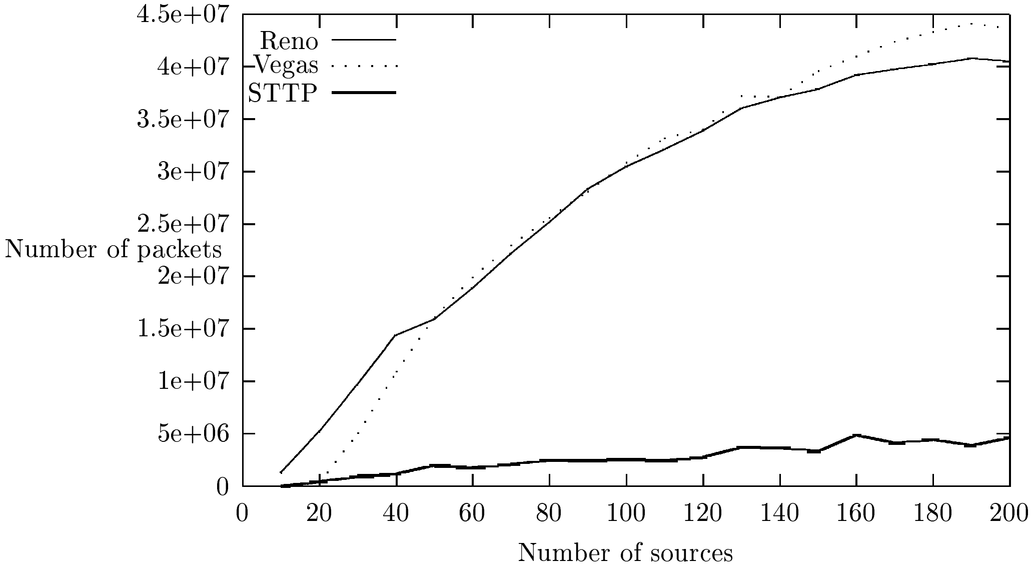

Figure 4.5: Experiment 2: TCP Reno, TCP Vegas and STTP Total Packet Loss

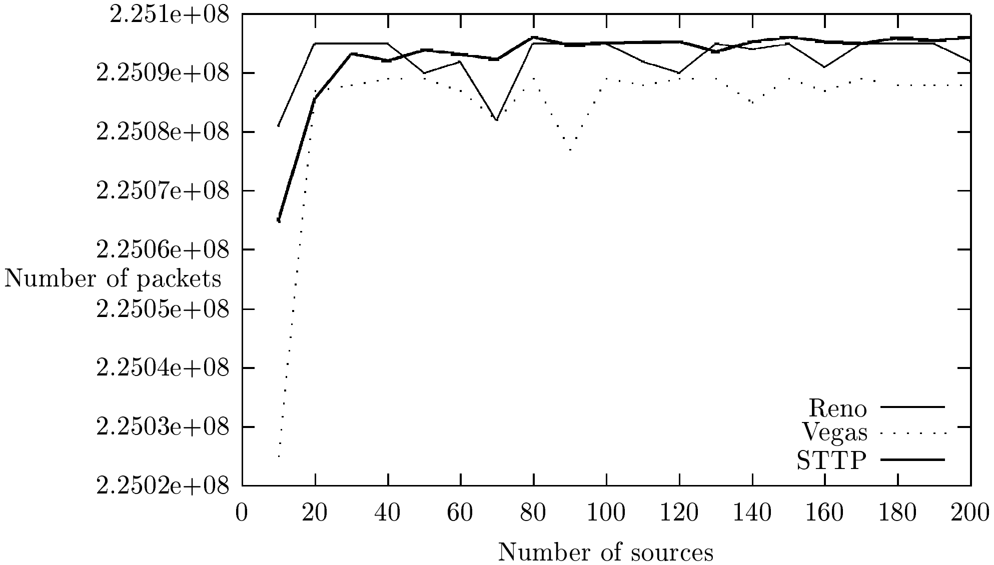

Figure 4.6: Experiment 2: TCP Reno, TCP Vegas and STTP Successfully Received Packets (Goodput)

standard TCP implementations (figure 4.6).

While the number of packets dropped (figure 4.5) is proportional to the number transmitted (figure 4.4), the goodput of each protocol (figure 4.6) is very similar above 20 sources. STTP, however, shows less variation than the TCP implementations.

In summary, the results from this experiment show that while the overall goodput between TCP Reno, TCP Vegas and STTP is similar, STTP achieves this standard of data transport with far lower packet transmission rates. This is a very positive attribute, and will aid the user-perceived quality of service by avoiding packet retransmission and by minimising the end-to-end delay through smaller packet queues at routers.

4.2.2 Experiment 3, Protocol Fairness of STTP

Traditional Internet transport protocols based on TCP Reno sense congestion through lost packets or excessive delay in the network. They then respond using multiplicative decrease in the congestion window. Normally, this results in a 50% reduction in the transmission rate in order to allow other competing streams to obtain their "fair share" of bandwidth. While this approach eventually attains the goal of fairness among streams, it is a lossy and often aggressive method, as shown in our experiments.

As the number of streams are incremented, lost packets in TCP Reno connections increase at an exponential rate as data is lost in order to accommodate new streams. As described in sections 2 and 3, STTP uses a simplified version of TCP Vegas's congestion avoidance algorithms. By monitoring increases and decreases in packets' Round Trip Time (RTT), it can sense pending congestion and so adjust its packet flow accordingly. This is done by proportional modification of the token bucket size and current token values. The result is much lower packet loss as can be seen in the above experiments.

However, the disadvantage of STTP's congestion avoidance mechanism is that it does not respond to new flows as quickly as that of TCP Reno. New TCP sources begin transmission with an exponential increase known as Slow Start. Existing TCP connections will continue to send at their current speed until difficulties are encountered and a reduction (50%) is made. New sources are then able to "grab" a larger share of bandwidth. STTP, on the other hand, does not experience such packet loss by aiming to avoid congested queues.

We will now assess the relative fairness of STTP compared with that of TCP Reno. Table 4.5 shows the results of further simulations conducted on the topology shown in figure 4.3. This time, simulations were run for 2000 seconds and all links were 128kb/s.

In order to highlight the effects of our history-based congestion avoidance algorithm, we have included simulations run on three different implementations of STTP. These are shown in figure 4.5 with "STTP α:β". When recalculating the bucket size and number of tokens in operation, α is the weight assigned to STTP's existing values. β is the weight assigned to the result of new calculations. For example, if a 15% decrease in RTT is detected, the following calculation is used: bucketValue = ((α * bucketValue) + (β * (1.15 * bucketValue))). The sum of α and β is 1.0 at all times.

By shifting more emphasis to β the protocol becomes more oriented towards the existing network state and will react more quickly to current events. However, by weighting the formula towards α, we obtain a more stable data flow which is not so quickly affected by new connections.

Table 4.5 shows results from a number of simulations using various weightings and it is evident that 5:5 or 1:9 ratios provide much better performance and fairness than more history-biased values. The fairness index laid out by Jain in [23], assigns a value between 0 and 1 with $Fairness = f_A(x) = [\sum_{i=1}^{n} x_i]^2 / \sum_{i=1}^{n} x_i^2 \, , x_i \geq 0$. Using this formula to process the results in table 4.5, we can see that for all experiments with more than a single source, TCP Reno yields an index of 0.99, as does STTP 5:5. STTP 1:9 gives 0.99 (2 sources) and 0.98 (4 sources), and STTP 9:1, 0.84 and 0.75 respectively. In this case, an index of 1.0 is totally fair and 0.0, totally unfair.

Traditional max-min fairness [14] states that given a set of limited network re-

Protocol	Source #	#packets transmitted	#packets dropped
1 Source			
TCP Reno	1	31980	0
STTP 9:1	1	31980	0
STTP 5:5	1	31980	0
STTP 1:9	1	31985	0
2 Sources			
TCP Reno	1	16093	0
	2	15924	0
STTP 9:1	1	22899	0
	2	9108	0
STTP 5:5	1	16084	0
	2	15918	0
STTP 1:9	1	16081	0
	2	15919	0
4 Sources			
TCP Reno	1	7773	77
	2	8156	73
	3	8244	67
	4	8124	70
STTP 9:1	1	13790	0
	2	5464	0
	3	6079	0
	4	6699	2
STTP 5:5	1	8181	0
	2	7146	0
	3	7943	0
	4	8754	0
STTP 1:9	1	7118	0
	2	6949	0
	3	9440	0
	4	8517	0

Table 4.5: STTP Fairness

sources, bandwidth should be shared as equally as possible between competing connections. At the same time, maximal usage of the available resources should be maintained.

Given the parameters of this simulation, the maximum number of 1000 byte packets which can be transmitted is 32,000. Table 4.5 shows that in single source simulations, both STTP and TCP Reno use around 99% of this capacity by successfully transmitting 31980 packets. In subsequent simulations, the link's resources are shared between a number of greedy FTP sources running over the relevant transport protocol. In some cases, the total number of packets transmitted exceeds 32,000, this is due to queueing which has taken place prior to sources being stopped at t=2000.

While neither TCP Reno, nor STTP conform precisely to max-min fairness, the results in table 4.5 show that through more aggressive, lossy flows, TCP Reno achieves more balanced flows. This is due to connections relinquishing large portions (50%) of bandwidth when data is lost and therefore allowing competing connections to expand their transmission rate. STTP exhibits significantly lower packet loss and so only balances its flows through variations in RTT.

From the above experiment, we have shown that given a number of streams, STTP will fully utilise the available bandwidth. Using a 9:1 ratio, it is not as quick to react to new traffic as TCP Reno, however, it does so fairly and with far fewer packet losses. The implication of this is that fewer segments of data would have to be retransmitted, and therefore give an improved perception of service quality to the user.

4.2.3 Experiment 4, Variable Bit-Rate Applications using NS

Our third set of experiments were conducted using the same build and STTP implementation as in previous sections. The network topology used was that depicted in figure 4.3, and used in Experiment 2.

A protocol's performance for constant bit-rate, greedy sources, such as FTP or SMTP transfers, is potentially different to that when dealing with variable bit-rate or interactive data streams. With a window-based flow control mechanism, it is possible for there to be delays in transmission of time-critical data (with the Nagle algorithms, for example [35]). STTP was designed with interactive and bursty data in mind, so facilitated by the use of token-bucket flow control, it is anticipated that its performance in data delivery and packet loss will be an improvement over existing TCP-like protocols.

In order to test STTP's performance with variable bit-rate applications, simulations were run in NS using the aforementioned network topology. A suite of simulations were run with between 10 and 100 traffic sources, each iteration increasing the number of sources by 10. Each traffic source took input in the form of a trace file, obtained from http://www.research.att.com/~breslau/vint/trace.html. This variable bit-rate stream is an NS trace of the Star Wars movie, and each stream is initialised at a random point within the trace file. Each iteration of the simulation is scheduled to run for 900 seconds, with sources being added at 2 second intervals. The resulting traffic statistics are shown in table 2 and figures 4.7, 4.8 and 4.9.

In our simulations with variable bit-rate traffic, STTP exhibits similar characteristics to those in previous, bulk transfer experiments. Both the number of packets transmitted and dropped is significantly lower than either TCP Reno or TCP Vegas. However, the goodput obtained is comparable, if not exceeding, that of TCP-based protocols (figure 4.9).

In all experiments, the measurements for transmitted and dropped packets were taken from core NS network monitors, not from within the protocol implementation itself. This gives an independent monitor of each protocol's performance without the possibility of distortion by a particular implementation's internal counters or algorithms.

In order to monitor each flow of data, a NS Flow Monitor was attached to the backbone, bottleneck connection (see figure 4.3), through which all traffic passes. For each traffic source, the Flow Monitor records transmitted and dropped packets.

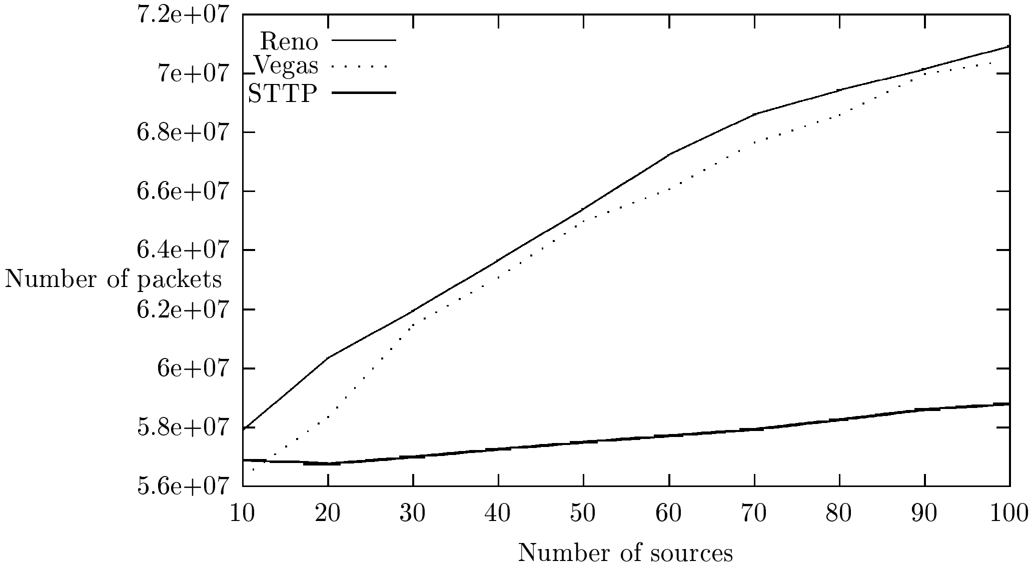

Figure 4.7: Experiment 4: TCP Reno, TCP Vegas and STTP Total Packet Transmissions

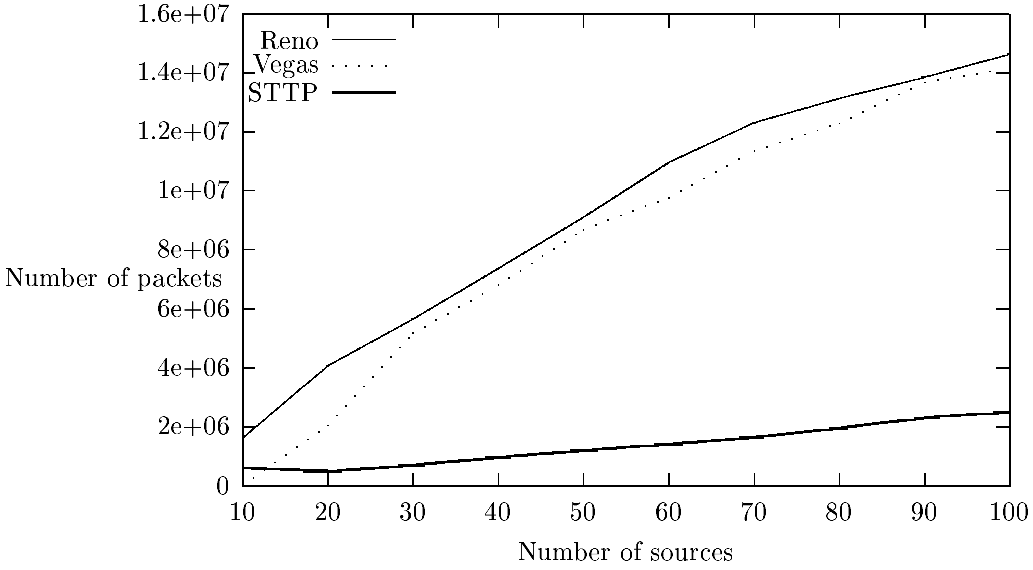

Figure 4.8: Experiment 4: TCP Reno, TCP Vegas and STTP Total Packet Loss

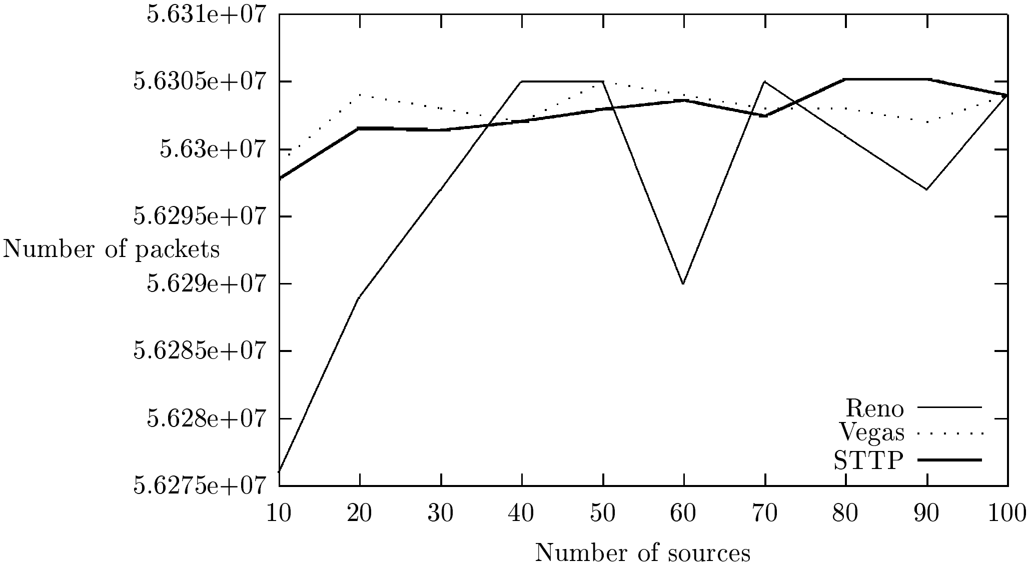

Figure 4.9: Experiment 4: TCP Reno, TCP Vegas and STTP Successfully Received Packets (Goodput)

Placing this monitor on the ingress ports to the bottleneck connection allows us to record the network activity of interest in these experiments. We were able to record the number of packets transmitted and dropped by individual protocol streams, and therefore construct the tables and graphs presented in this thesis.

4.2.4 Experiment 5, HTTP Applications using NS

This set of simulation experiments with STTP in the NS simulator was conducted using a tracefile of real HTTP traffic from the University of Leeds Virtual Science Park servers. These HTTP servers offer an interactive Web interface to a relational database. For further information, visit the project's Web pages at http://www.vsp.co.uk.

In order to gather data, a machine was connected via a hub to the live VSP HTTP server. This allowed us to capture all TCP socket port 80 requests arriving at the server using 'tcpdump' for a three day mid-week period. Once captured, we filtered the trace file to give only incoming requests. A script was then written to capture

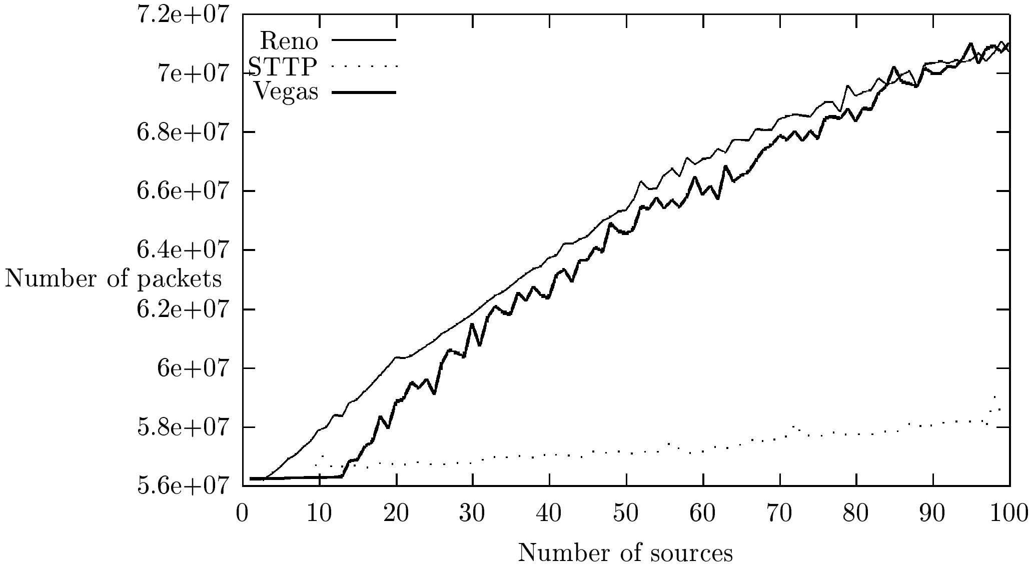

Figure 4.10: Experiment 5: TCP Reno, TCP Vegas and STTP Total Transmitted Packets

only the packet interarrival gap and request size. This data was then encoded in NS tracefile format and used as input for the traffic sources in this experiment. For the purposes of this experiment, only the packet size and their inter-arrival gaps were required. The data was therefore anonymous.

The bursty nature of interactive HTTP traffic is typical of present Internet applications, and allows us to test both congestion avoidance mechanisms in addition to the behaviour of STTP's token-bucket flow control.

The network topology and simulation configuration are identical to that in Experiment 4, with suites of simulations being run with between 10 and 100 sources, for a duration of 900 seconds. Each traffic source begins its transmission at a random point within the HTTP trace file.

Figure 4.10 shows the number of packets transmitted for each protocol; Figure 4.11 shows the number of packets lost due to network congestion; and Figure 4.12 shows the difference in terms of data goodput.

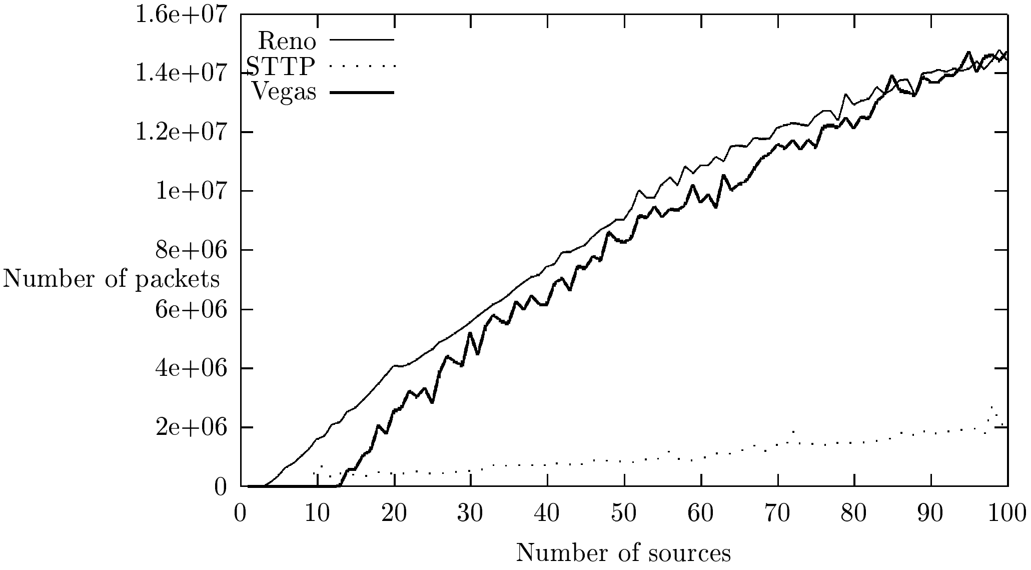

Figure 4.11: Experiment 5: TCP Reno, TCP Vegas and STTP Total Packet Loss

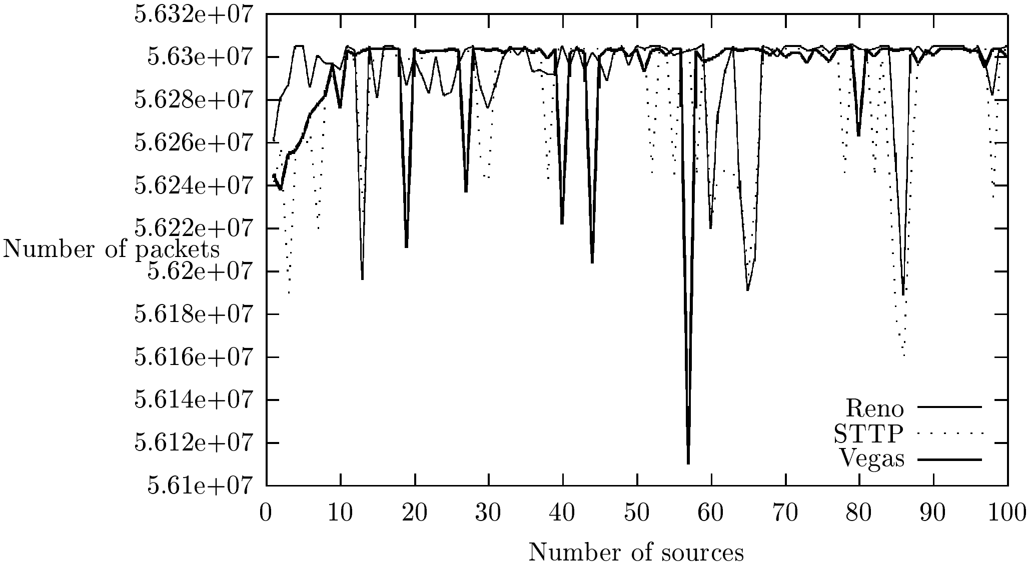

Figure 4.12: Experiment 5: TCP Reno, TCP Vegas and STTP Successfully Received Packets (Goodput)

4.2.5 Experiment 6, Mixed Simulation of TCP Reno, TCP Vegas and STTP

In this simulation we compare the performance of TCP Reno, TCP Vegas and STTP running simultaneously on a given network with variable-rate data sources for transportation.

Experiment 6 was conducted using the topology depicted in figure 4.3, as used for other experiments in this chapter. The simulation was run with between 3 and 99 transmitting / receiving network nodes. The bottleneck link between routers on the network was set at 100Mb/s with 1ms latency. Local connections to both transmitting and receiving nodes were configured at 10Mb/s with 1ms latency.

Transmitting nodes were equally divided between TCP Reno, TCP Vegas and STTP (33 nodes running each protocol) and configured sequentially. The simulation script started each node in turn, two seconds apart, thus giving a roughly balanced profile of transmitting sources. The entire simulation ran for 900 seconds before terminating. All sources were therefore active 200 seconds into the experiment. The traffic source used for each transmitting node was the live HTTP trace file, obtained from the University of Leeds Virtual Science Park Web servers. This traffic type would give a bursty profile, and would test the startup and congestion avoidance algorithms of each protocol.

This experiments link speeds and latencies were selected to approximate the bandwidth available to LAN users with campus area connectivity, and is designed to test high speed medium-area network performance. In particular, this test allows us to observe the performance of TCP Reno, TCP Vegas and STTP in states of normal flow, and congestion. With fewer than 10 sources transmitting at full line speed, it is not possible to fill the bottleneck network segment. Each transmitting and receiving node is connected to the bottleneck link with a 10Mb/s link. Therefore, congestion can only occur when simulating more than 10 nodes. Furthermore, when congestion does occur, it will not be as heavy as in previous experiments, due to the increase bottleneck link capacity. This is designed to test the interaction between congestion

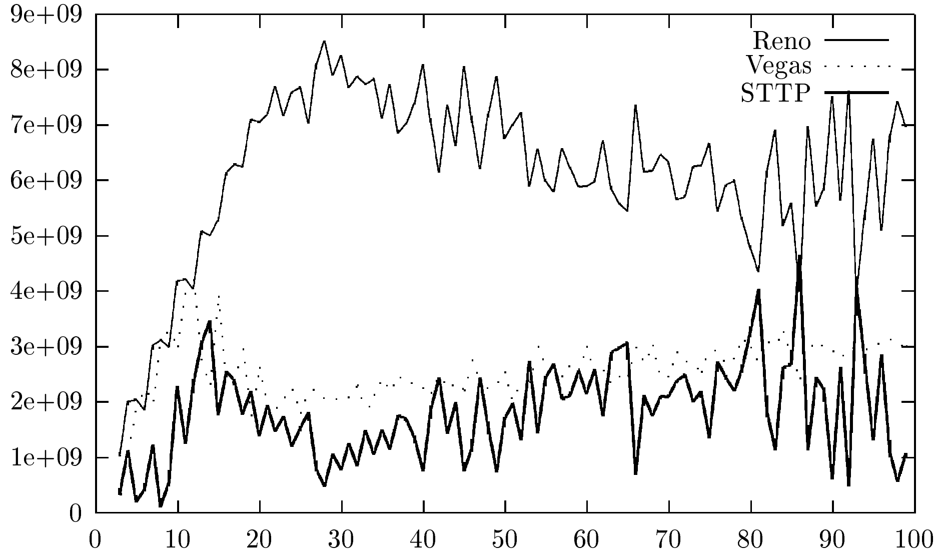

Figure 4.13: Experiment 6: TCP Reno, TCP Vegas and STTP Successfully Received Packets in Mixed Simulation

avoidance algorithms as opposed to simply the way in which each protocol will react to extreme congestion scenarios.

The goodput results for TCP Reno, TCP Vegas and STTP in this simulation scenario are presented in figure 4.13. Initial performance figures for TCP Reno and Vegas with between 3 and 12 sources are very similar. It is only when TCP Reno's congestion avoidance algorithms are competing with delay-sensitive ones in TCP Vegas and STTP, that TCP Reno begins to obtain significantly higher packet goodput.

Both TCP Vegas and STTP will proactively reduce their transmission rate if network congestion is sensed. TCP Reno, on the other hand, will only reduce its transmission rate if packets are lost, or incur excessive delay. Therefore, TCP Reno will continue to increase its transmission rate at the expense of both TCP Vegas and STTP. As can be seen from figure 4.13, TCP Vegas and STTP follow a similar trend in their packet goodput as the number of sources increases. Throughout the experiment, packet loss is very low, at less and 1%, which gives goodput which is roughly equal to the number of packets transmitted by a given source.

This simulation shows us that proactive congestion avoidance mechanisms are unlikely to obtain a fair share of the available bandwidth on a given link when working alongside TCP Reno, or other legacy IP transport protocols such as TCP Tahoe.

In order to counteract the aggressive nature of TCP Reno's congestion avoidance algorithms, the parameters, as discussed in section 4.2.2, could be adjusted. However, this leaves the proactive protocol in a situation similar to that of TCP Reno, in which it will behave in an aggressive manner towards delay-sensitive algorithms. We therefore consider that future work in the area of intelligent congestion avoidance algorithms would be most beneficial to this thread of research. This topic is further outlined in section 6.3.

4.3 Analysis

Throughout our simulation experiments, STTP has been consistent in its performance. In addition to highlighting key features of the experimental protocol, this consistency and conformance with our algorithmic specification also aids the validation of the simulation packages themselves. This chapter has seen the simulation testing of STTP in comparison with TCP Reno and Vegas with a variety of traffic types. This fulfills objective K4 of our thesis objectives from chapter 1.

Experiment 1 in section 4.1.1 showed that in basic functionality testing, STTP exhibited zero packet loss and smoother packet transmission, compared with TCP Reno, which incurred significant packet loss with even a single data-flow. Table 4.4 shows how STTP's pro-active congestion avoidance was able to decrease variation in its transmission rate (RTT column). Here, we can see that the standard deviation (sd) for STTP sources is up to 50% of that for TCP Reno in the same scenario.

Experiment 2 showed that STTP transmits up to 20% fewer packets (figure 4.4 and exhibits up to 10% less packet loss (figure 4.5) than TCP Reno and TCP Vegas in bulk data transfer simulations. Figure 4.6, however, shows that the goodput for STTP in this experiment is within 1% of TCP Reno and TCP Vegas. This experiment demonstrates that STTP is more efficient than TCP Reno or TCP Vegas, as it is able to achieve a comparable rate of goodput but with the transmitting node having to retransmit significantly fewer packets.

Experiment 3 demonstrated that given a number of competing data streams, STTP will fully utilise the available bandwidth on a network path. We ran individual simulations with STTP, TCP Reno and TCP Vegas streams, which competed for bandwidth on a bottleneck link. While no protocols entirely conformed to traditional max-min fairness, we calculated that the fairness index for TCP Reno and STTP was 0.99. An index value of 1.0 is totally fair, and 0.0 totally unfair.

Experiment 4 presented data from simulations with STTP, TCP Reno and TCP Vegas, using variable bit-rate traffic sources. In this simulation, STTP again transmitted up to 20% fewer packets, with up to 10% fewer packets being dropped in the network. The goodput graph (figure 4.9 for this simulation showed that all protocols achieved the same (within 1%) level of goodput. From this information, we concur that STTP is able to maintain its performance with variable bit-rate traffic sources as seen with bulk data in experiment 2.

Experiment 5 used HTTP trace files to run individual simulations with TCP Reno, TCP Vegas and STTP. The results show that all protocols achieve the same levels of goodput (again within 1%), but this time with comparable levels of packet transmission and loss.

Experiment 6 ran combined simulations with TCP Reno, TCP Vegas and STTP being used simultaneously on the same network topology. We discovered that both TCP Vegas and STTP were adversely affected by TCP Reno's aggressive congestion avoidance algorithms. This experiment showed (figure 4.13) that proactive congestion avoidance algorithms need specific tuning in order to compete with legacy, aggressive protocols.

When compared with TCP Reno and Vegas variants, we noted several major improvements in performance. These benefits are related to key area K3, 3a, 3b and 3c, as described in chapter 1.

- Goodput comparable to that of TCP variants

- Significantly lower packet loss
- Lower number of transmitted packets
- Lower RTT and deviation measurements

4.4 Summary

The goodput of a connection, i.e. the number of transmitted packets which successfully arrive at their destination, is an accepted metric for the performance evaluation of a transport protocol. In the section on Fairness (section 4.5), we showed how STTP's parameters can be modified in order to make it more or less aggressive when reacting to network congestion. While an aggressive protocol may initialise its connections at a higher rate and be able to take advantage of newly available bandwidth on a given path, it is also more susceptible to packet loss. In comparison with our standard STTP model, used in our main experiments, TCP Reno and Vegas can be considered to be aggressive transport protocols.

In further experiments with varied traffic types, TCP Reno and Vegas were shown to achieve a similar overall level of packet goodput to STTP, while having increased packet transmissions and loss. In a real world implementation, this would yield greater CPU utilisation by the receiving hosts, as they would have to contend with re-ordering incoming packets as lost data was retransmitted. The transmitting host would incur additional overhead through increased packet transmissions.

Chapter 5 will go on to discuss our simulation results in greater detail.

Chapter 5

Discussion and Evaluation

In this chapter, we will discuss the results from our simulation experiments in chapter 3. Each suite of experiments from functionality and performance testing are considered in turn with focus being placed on metrics such as packet loss, overall packet goodput, and fairness.

Experiment #	Demonstrates
1 (section 4.1.1)	The functionality of STTP's congestion avoidance algorithms and flow control
2 (section 4.2.1)	The performance of STTP with bulk data transfers
3 (section 4.2.2)	The fairness of STTP in comparison with TCP Reno and TCP Vegas
4 (section 4.2.3)	STTP and TCP performance with Variable Bit-Rate video sources
5 (section 4.2.4)	STTP and TCP performance with bursty HTTP traffic sources
6 (section 4.2.5)	The performance of STTP, TCP Reno and TCP Vegas in a mixed protocol environment

Table 5.1: Experiment Information

Table 5.1 shows the experiments conducted in chapter 3. Our experiments were centered around two main types of network traffic. Firstly, traditional Internet applications, such as FTP, generally transfer large quantities of data in a single bulk transfer. A separate FTP transfer is initiated for each file requested and generally lasts for seconds rather than minutes. Secondly, modern applications, such as the World Wide Web (WWW), can use HTTP to transfer WWW page components from an Internet server to the viewing client. A given page may consist of many individual, small, components. User interaction during a WWW session will therefore impact the network with bursty transfer of many page components. Furthermore, the arrival of broadband access means that Internet users are likely to view an increasing amount of multimedia content. In particular, movie and informational data (news broadcasts, for example) form an increasing percentage of Internet traffic.

Our simulation experiments therefore use three traffic models. In order to address traditional Internet traffic requirements, a *bulk data* transfer model (FTP) was used. This model simulates a greedy traffic source, one which always has data to transmit. Data will be transmitted as quickly as possible, and will not end until explicitly instructed to do so by the simulator. This traffic model was used in experiments 1,2 and 3. A *HTTP tracefile* was used in experiments 5 and 6 in order to accurately represent HTTP transfers over the Internet. The contents of this tracefile were obtained using the method described in section 4.2.4 of chapter 3. An *MPEG tracefile* was used to simulate extended multimedia streams being carried by a reliable transport mechanism. The source of this tracefile is given in section 4.2.3. Experiment 4 examines the performance of TCP Reno, TCP Vegas and STTP transporting bursty, variable bit-rate data.

In order to address K4 and C3 (as detailed in chapter 1), the above simulations were carried out using bulk data and bursty traffic models. In this manner, we were able to demonstrate the effectiveness of STTP in comparison with current implementations of TCP.

In section 5.1, we discuss the results of our experiments (C4), which is summarised in subsection 5.2.5. Section 5.2 presents lessons learned during the course of our work, and general observations on protocol development.

5.1 Discussion of Simulation Results

In sustained FTP transfers, simulations of STTP showed that overall link utilisation was at least as good as that of TCP. In most cases, STTP made better use of the available resources due to its fast startup model, and packet-pair probing. The only cases in which the prototype model of STTP was not successful in obtaining a fair share of network bandwidth was when competing with aggressive congestion avoidance algorithms, used in TCP Reno, or in cases of extreme congestion. The latter case posed a problem for STTP because network probe packets would be lost, and the prototype model was not programmed to re-attempt connection setup. The ability to discover available bandwidth and commence transmission at a suitable rate meant that in a given simulation period, STTP could transmit several more packets than TCP by this method alone.

Further advantages to STTP's transmission profile were gained by its proactive congestion avoidance algorithms. Whereas TCP Reno relies on aggressive transmission and packet loss to discover congestion, STTP was able to reduce its rate before such losses occurred. This resulted in significantly lower packet loss ratios for all the experiments conducted in the course of our research. In section 5.2, we discuss how it is possible to modify STTP's parameters to give an aggressive, lossy protocol, similar to TCP Reno.

Due to STTP's default congestion avoidance parameters being weighted towards minimising queue lengths, the round trip times experienced in its conversations were notably shorter than those of TCP Reno. The reason for this is that STTP will decrease its transmission rate if there is a significant increase in the measured RTT's. By doing so, it prevents long queues from building up and hence reduces the likelihood of packet loss. Conversely, TCP Reno will transmit at an incremental rate until packets are lost, or timeouts occur.

The result of this behaviour is that TCP Reno often experiences packet loss and leads to the famous "sawtooth" traffic profile of a stable TCP source. In this state, traditional TCP Reno will perform linear increments to its congestion to the point of packet loss or timeout. At this point, it reduces its window by 50% and recommences linear congestion avoidance. It has been shown in many publications that the "linear increase, multiplicative decrease" algorithm is both fair and stable.

The sawtooth behaviour of TCP Reno affects the RTT of its own, and competing conversations. As incremental congestion avoidance advances, the queue length at a given router on a conversation's path will increase, hence steadily increasing the RTT. When a conversation reaches its point of congestion (and data is lost), its transmission rate is halved, resulting in a drastic reduction in the number of packets queued at any one time. With fewer packets now queued, the RTT for conversations using a given router will now drop.

5.1.1 Functionality Testing with the REAL Simulator

Figure 4.4 shows this effect in the standard deviation times for various TCP and STTP conversations. In every case, the standard deviation for STTP connections is significantly lower than that of TCP Reno under the same conditions. Furthermore, STTP exhibited zero packet loss in this experiment.

The simulation topology used in Experiment 1 was designed to test a protocol's reaction to congested, bottleneck links. By adding sources at regular intervals, we were able to increase the load on routers and network links, thus adding to existing queues. As further conversations were added during heavy congestion, we also tested the startup algorithms for each protocol. A final objective was to measure the relative bandwidth allocation for each conversation during the simulation.

While first impressions may be that TCP has successfully transmitted many more packets during the simulation than STTP, the number of packets dropped must also be taken into account. For example, for conversation 1 of table 4.4, TCP Reno transmits on average 35.56 packets, and STTP 37.89. On average, 24.89 of TCP Reno's packets were dropped by the network. STTP dropped 0 packets. Similarly, in the later conversations, TCP loses a significant number of its transmitted packets through poor bandwidth management. STTP transmits fewer packets but has a significantly higher overall *goodput*.

The behaviour of STTP in this case shows better management of available network resources and greater consideration to competing data streams. Fewer packet drops and retransmissions would lead to an improved Quality of Service to the end user, as a more constant flow of data is maintained. Furthermore, such a stable connection would allow easier management of the network by its administration as if scaled accordingly, traffic as a whole would be much less bursty.

5.1.2 Performance Testing with the NS Simulator

In our experiments with NS, we examined the performance of STTP and TCP Reno/Vegas. STTP continued to show many good characteristics such as dropping fewer packets and achieving higher rates of goodput. An important feature of our proactive congestion avoidance algorithm was also highlighted.

Once established, STTP uses variation in RTT to detect congestion. If packets are lost then it reacts just as TCP Reno. However, with new STTP connections probing for available bandwidth and setting their transmission rate accordingly, our results show that fair allocation of bandwidth is not achieved as quickly as TCP Reno.

The reason for this is STTP's startup mechanism. TCP Reno adopts a lossy, aggressive approach with its slow start algorithm. This has been identified as causing the majority of TCP's packet loss during the lifetime of a connection [47]. As TCP expands its window (exponentially), other competing sources are forced to drop packets and back off. This could be by 50% through congestion avoidance, or to a full slow start if timeout occurs. This allows the new sources to start up and obtain a share of the newly available bandwidth. While this technique is more conducive to short-term bandwidth sharing, our results have shown it to be both bursty and lossy.

Tables 5.2 and 5.3 present coarse grained results for two experiments, which were run in NS. There were 40 FTP traffic sources, traveling over 10Mb, 1ms links. The simulation durations were 200 and 300 seconds accordingly with sources starting at five second intervals. The network topology used for these experiments is the same as that depicted in figure 4.3. Each table shows the number of packets transmitted and dropped, grouped by protocol.

Table 5.2: Long Duration (200 seconds) STTP and TCP Reno - 40 sources

protocol	duration	bytes Tx	bytes dropped	goodput
reno	199.900	22407000	2613000	19794000
sttp	199.900	21187120	1429640	19757480

Table 5.3: Long Duration (300 seconds) STTP and TCP Reno - 40 sources

protocol	duration	bytes Tx	bytes dropped	goodput
reno	299.900	33244000	3451000	29793000
sttp	299.900	30167600	416240	29751360

The tables show that TCP has a significantly higher rate of packet loss when compared with STTP and therefore a higher rate of retransmission. Given this, we can conclude that STTP will make more effective use of available bandwidth in congested networks by detecting congestion before packet loss takes place. It is important, however, to examine how each individual stream performs.

In section 4.5 [50], we examine the issue of fairness more closely. The results of running more lengthy simulations with competing sources were as anticipated. Given time to stabilise, STTP achieves levels of fairness comparable with that of TCP Reno in the same environment. This process can be made faster through modification of STTP's internal variables, which will be discussed further in subsection 5.2.3, and later in section 6.

To conclude the simulation of STTP and its comparison with TCP Reno and TCP Vegas, a range of simulations was run with all three protocols on the same topology, while varying the number of sources from 10 to 200. Different traffic types were used; bulk transfer, variable bit-rate trace file, and HTTP request trace file.

In Appendix table 1, it is evident that both TCP Reno and TCP Vegas transmit a far greater number of packets than STTP. However, given the data shown in figure

4.5, we can see that a relatively high percentage of this is dropped. The difference between the number of packets transmitted and the number of packets dropped, is termed the "goodput" of a connection. This indicates how many packets successfully arrived at their destination. This is shown graphically in figure 4.6.

In all cases, STTP transmitted and dropped significantly fewer packets during the course of the simulations. It did, however achieve a comparable level of goodput. This indicates that STTP is able to transmit just the right amount of data to keep a bottleneck connection full, while not being too ambitious. The advantages of this approach are numerous and include; STTP does not load routers with aggressive packet transmission, applications using STTP will be given a better Quality of Service due to fewer transport layer frames having to be retransmitted, computationally expensive tasks such as frame retransmission are less frequent than with existing TCP implementations.

In figures 4.6, 4.9 and 4.12, STTP is shown to have goodput comparable with that of TCP Reno and TCP Vegas. In many iterations of our simulation, it achieves the highest goodput of all the tested protocols. This result should be considered in conjunction with the graphs showing dropped packets, figure 4.5, 4.8 and 4.11. Given that the goodput of all protocols is at least similar, STTP incurs far fewer packet drops than other protocols. The result of this is that the available bandwidth is used much more efficiently by STTP streams. We can therefore conclude that the algorithmic framework adopted by STTP is more efficient than that used by existing standard TCP implementations. The framework within which STTP was developed adds to our confidence in these results. The design of NS is highly modular. In order to implement STTP, only the algorithms directly related to the transport layer were modified. In order to aid this design, the NS TCP Reno model was modified to incorporate STTP's algorithms. The rest of the simulator was left untouched.

A fundamental concern with flow controlled transport protocols is to ensure that they make good use of available bandwidth. With the advent of Virtual Private Networks, where bandwidth is often reserved and guaranteed, a protocol should be able to quickly utilise available resources. However, it is also important to share these resources efficiently between competing connections.

Unfortunately, as the number of sources increases, the inability of the Packet-Pair probe to allocate bandwidth in a congested environment is highlighted. With more than 40 sources in operation, our detailed traces show that certain STTP flows were only able to transmit two probe packets of 40 bytes. In cases of high congestion, either a probe packet is lost, or the result yields such a low bandwidth that the token bucket cannot be initialised with even a single full-size token. This behaviour is also noted in TCP Reno where, as the number of sources increases, certain ones are not able to successfully establish a connection with the remote host.

In our experimental implementation, STTP is not programmed to re-attempt connection in these cases, so such flows do not succeed in connecting to their destination host. There are several potential solutions to this problem. Firstly a Packet-Pair train could be used, as discussed in [1]. This would reduce the probability of probe packet loss and also give a more accurate estimation of the available bandwidth. Secondly, a fast retransmission mechanism could be formulated for Packet-Pair probing, which would re-attempt connection setup after a short timeout period. Probe packet responses should really be acknowledged within one second except in cases of extremely high latency. Furthermore, a retransmission of probe data would not significantly impact network performance and could be used to realign token bucket settings when received, even if previous attempts proved to be successful, but with high latency.

5.2 Evaluation and Lessons

In this section we will discuss our findings, in particular how they relate to specific techniques we have chosen to deploy in our experiments.

5.2.1 Protocol Performance

In designing STTP, we have produced a protocol which exhibits very low packet loss, and timely, guaranteed delivery of application data. The results in chapter 3 show that it is capable of such performance even when transporting bursty, variable bit-rate data.

Interactive network applications generally consist of small user requests, which need to be delivered quickly and without unnecessary retransmission. A protocol with higher levels of packet loss (and therefore retransmission) will affect the user-perceived quality of service to a greater extent, depending on the regularity and burst size of loss periods. Irregular, variable-length packet loss will degradate the performance of an interactive network application to a point where it becomes unusable.

From our experimental results, we can therefore conclude that the protocol model adopted by STTP will facilitate the delivery of such applications. There are, however, some lessons to be learned from the implementation of Packet Pair probing and Pro-active congestion avoidance algorithms.

5.2.2 Packet Pair in Congested Networks

The Packet Pair bandwidth probing technique proved to be reliable in our simulations for discovering the available bandwidth on a network path. Even in scenarios where a new connection was probing into an already congested network, this technique produced accurate reports.

When congestion is heavy, however, there are issues with our approach to protocol startup. STTP may only be able to probe a small amount of bandwidth available on a network path. This means that the initial bucket size for the connection is minimal, and the protocol will depend on variations in RTT to increase its transmission rate. However, as a new source has entered the path, the increased queue length should be sufficient to trigger a reduction in competing traffic. The fairness of such a model is discussed further in [11] [34] [7]. Our work in this area was reported in section 4.5.

5.2.3 Aggressive vs Timid Sources – the fine line

Given the range of tunable parameters in STTP, it can take on a variety of characteristics. By adjusting its sensitivity to variation in RTT, the protocol can be made more or less aggressive to competing traffic sources. This is achieved using the α and β parameters of STTP's congestion avoidance algorithm. In order to make STTP more sensitive to increases in RTT, the difference between these values should be reduced around 1.0.

Simulation experiments were conducted in [51] using these parameters, where the difference in behaviour between aggressive and timid sources was noted. As α is decreased, so STTP's sensitivity to decreased RTT is reduced. The effect of this is that the protocol is slower to react to newly available bandwidth. This is due to the RTT having to reduce by a more significant amount before STTP will increase its mean transmission rate. Conversely, as β is increased, so STTP becomes less sensitive to increases in RTT and slower to relinquish bandwidth to new or expanding connections.

The result of adjustments as described above, is a very strong transport protocol which, once established, will defend its bandwidth share. Unfortunately, this is not conducive to equal and fair performance between competing streams. Papers referenced in chapter 2, section 2.2.2, discuss potential solutions to unfair behaviour, and alternative implementations of TCP's congestion control.

At the other extreme, a small delta between α and β can be selected. The effect of this would be to have STTP react quickly to both increases and decreases in RTT. Unfortunately, our experiments show that this results in bursty and unpredictable behaviour which, while this may be in the favour of competing streams, does not facilitate the QoS for the application being served.

A further point particular to STTP, is that the degree to which the protocol is sensitive to decreases in RTT is directly related to the rate at which it will increase its rate of transmission. It is therefore of great importance that appropriate values for α and β be selected.

During the course of our work, we have conducted many hundreds of simulations with STTP over a broad range of network topologies. The static settings we have chosen to use have proven to be reliable in our scenarios, and to deliver satisfactory performance, delivering packet goodput comparable with that of TCP Reno and TCP Vegas. We do, however, believe that there remains a great deal of interesting and highly useful work to be conducted in this area. Section 6 covers this in more detail.

5.2.4 Discussion of Software Simulation and Prototyping

During the course of our experimentation, areas were covered which provided useful insight into the issues surrounding protocol development. In certain cases, the problems encountered were particular to a given simulation package, but others were more general in nature.

A common issue with the simulation packages used for this research was that despite normally reliable behaviour, certain scenarios, topologies, or protocol mixes, would result in early termination or crashing of the software. A common issue with both REAL and NS was for simulations of extended duration with a large number of sources to become unstable. It became evident that certain high speed topologies would not run for extended periods without terminating abnormally. This behaviour was, however, confined to a small number of cases.

Fortunately, we were able to find stable parameters for more lengthy simulation runs and used them accordingly. This allowed us to test protocols under a variety of environments, not only short and medium length, low bandwidth scenarios.

Conversely, when performing a kernel implementation of our TCP modifications [49], it became difficult to sufficiently stress the local area network (10Mb/s). Our aim was to force TCP to timeout and restart its connections. However, with such a large amount of bandwidth and relatively low latencies, it was necessary to use lower capacity modem links to examine its behaviour more closely. We were subsequently able to induce connection restarts on the LAN implementation through further kernel enhancements.

This approach was successful, and we were able to test the functionality of our modifications and to see that they performed according to our models. Unfortunately, without more extensive equipment and resources, it was not possible to exhaustively test them in a high speed LAN environment.

5.2.5 Summary of Simulation Experiments

In the above discussion, we have seen how the functionality of STTP was proven using Keshav's REAL simulation package and compared with TCP Vegas using bulk data flows. We concluded from these tests that the STTP framework was a viable research project, and that the second stage of performance testing should be pursued.

An implementation of STTP in NS allowed us to perform more detailed testing with a wider variety of application traffic types and transport protocols. The results seen in REAL were supported by those from NS and further enhanced with fine-grained, large-scale simulation experiments. Lengthy simulations which compared STTP, TCP Reno and TCP Vegas highlighted key features of our congestion avoidance algorithms and start-up model. We saw that a pro-active congestion avoidance model can be highly effective in minimising packet loss while maintaining the overall rate of transmission. The result is a transport protocol which yields significantly higher goodput than existing Internet layer four implementations.

Key areas K3 and K4, as described in chapter 1, outline the aims of our protocol design. The simulation results (providing C3) show that we have developed a transport layer protocol which performs at least as well as existing TCP implementations. Experiments 4 and 5 demonstrate STTP's ability to deliver comparable levels of goodput (within 1% of existing TCPs) but with up to 20% fewer packets being transmitted, and up to 10% fewer packets being dropped in the network. STTP has been simulated with a key set of traffic models, with as much data as possible being based on live traffic traces. This methodology gives sound foundation to the results of chapter 3 (C3).

The above study and discussion of our experimental results conclude thesis contribution C4.

Chapter 6

Conclusions and Future Work

6.1 Summary

In this thesis, we have presented our work on the examination, design, modelling, implementation and simulation of STTP, an alternative transport protocol. We have studied the currently de facto standard of TCP (K1), and performed practical experimentation with the Reno and Vegas TCP variants. We have identified areas in which current TCP congestion avoidance algorithms could be improved (K2) and produced prototype modifications to TCP Reno (C1 in chapter 3). C2, the design of a replacement transport protocol is presented in chapter 3. This contribution covers key area K3. The subsequent simulation of our prototype is presented in chapter 4 and realises contribution C4.

Our simulation experiments have shown how a pro-active congestion avoidance model may be more appropriate for use with traditional, or bursty, network applications than current de-facto standards such as TCP Reno or TCP Vegas. Our solutions, which utilise packet-pair startup techniques, pro-active congestion avoidance, and token bucket flow control, have shown themselves to give performance comparable with that of current TCP variants. However, the number of packets dropped and retransmitted is significantly reduced using pro-active congestion avoidance mechanisms. Furthermore, the use of token-bucket flow control allows applications to send bounded bursts of data while maintaining an overall mean rate, which is in accordance with the currently estimated bottleneck bandwidth.

The implication of our simulation results for interactive network traffic is to facilitate user-perceived Quality of Service in addition to making more efficient use of available network resources. A transport protocol, which will deliver data in a timely manner with a low degree of packet loss, is able to provide the application layer with smooth inter-host communication. From the user's perspective, the arrival of information becomes more predictable and the application therefore more usable.

If immediately successful, a simple packet pair probe should allow the application to open a socket and burst data up to the capacity of the bottleneck link. In contrast with TCP Reno's Slow Start algorithm, this process should greatly aid brief interactive sessions, such as those seen with Web-based services. Experiment 5 in chapter 4 demonstrates the effectiveness of STTP in this scenario.

6.2 Contributions

The framework used in the design of STTP employs proven components from a variety of sources. The problem space associated with existing TCP algorithms when transporting bursty or short-lived data streams, was broken down into three key areas (K3).

- Connection initialisation and startup
- Congestion avoidance and control
- Packet loss and recovery

Each of these was addressed in turn with the Packet-Pair startup, Token Bucket flow control and Pro-active congestion avoidance algorithms. The functionality testing, executed in chapter 3, illustrated the interaction of these algorithms and how this modular design was able to operate in line with existing TCP implementations. Further testing was then carried out in a more advanced simulation environment with varied traffic types and congestion scenarios. Throughout these experiments, the advantages of the STTP framework was consistent with our projected model.

The resulting experimental protocol exhibits packet goodput comparable with existing TCP implementations, transporting a variety of traffic types. Its fundamental advantages are that significantly fewer packets are transmitted and dropped due to the improved protocol framework. The available bottleneck bandwidth is probed and discovered in a single round-trip, and active data transmission commences at an appropriate rate. Congestion is then sensed pro-actively, which helps avoid network congestion, minimise router queues, and relieve packet loss and retransmission.

The main contributions of this work are therefore:

[K1] the survey of existing protocol research in order to identify key problem areas

[K2] the evaluation of TCP modifications as a potential solution

[K3] the design and simulation of an experimental protocol, which deploys proven techniques in bandwidth discovery, flow control, and congestion avoidance

1. Connection initialisation and startup
2. Congestion avoidance and control
3. Packet loss and recovery

[K4] evaluation of the experimental protocol against existing TCP implementations with a variety of network traffic models

Original contribution was made through modifications to the existing TCP Reno implementation (referred to as C1) and with an experimental protocol (C2), which has been designed specifically to address the issues mentioned above. C1 is addressed in section 3.1, and C2 throughout chapter 3. Simulation of the resulting protocol is carried out in chapter 4 (C3). Further detailed analysis of our protocol simulations can be found in chapter 5 (C4).

6.3 Future Work

The results reported here open up several further avenues for investigation. The techniques employed by TCP Vegas and other pro-active transport protocols often employ hard-coded (or set) variables for RTT or throughput measurements. Given the highly dynamic nature of network traffic, and the rapid evolution of physical layer technologies, we feel that an active approach to congestion avoidance is needed. In order to provide a solution which will scale with future developments, indications from our experiments suggest that a self-modifying approach may be required in order to yield optimal performance.

If the delta of α and β is small around 1.0, then STTP becomes highly responsive to fluctuations in RTT. Conversely, as α decreases and β increases, a more stable yet stubborn behaviour is produced. It would be our intention that the protocol be able to modify these parameters in run-time according to a given set of heuristics. For example, should a stream's transmission rate being to fluctuate rapidly, then the delta may be increased in order to stabilise the current connection. Alternatively, if a connection senses that it is becoming ”squashed” by competing traffic, it may become more aggressive in order to sustain the current level of QoS for an application. Such responses are likely to significantly benefit user-perceived QoS.

It has also become apparent during the course of this work, that the rigid separation between traditional OSI layers may not provide the most efficient means of communication between Internet hosts. The advent of ATM has shown how duplication of effort at different layers in the stack can produce conflicting results. There is, however, ongoing research which suggests that communication between ATM's congestion avoidance algorithms and TCP will resolve such issues [15].

The suggestion that open, direct communication between layers should take place is clearly ludicrous, as this may well lead to chaotic behaviour due to conflicting information. While the network layer may believe there to be available bandwidth on a connection, the application may wish to reduce transmission rates for QoS management.

The solution ultimately falls to kernel design and the provision of an appropriate API for application and hardware driver developers. If a shared area of memory were allocated for the presentation and retrieval of QoS and network information, the various layers could obtain the data required in order to govern their transmission rates and offered QoS. Similarly, they could display their own variable data in order to inform other layers of their performance and measurements.

Explicit communication has been show to be effective in work conducted by Floyd [18] and we believe that such a framework, if implemented on a local host, could be used to great effect both for current connections and in the initialisation of new streams.

In order to deliver end-to-end QoS for interactive network applications, it is becoming increasingly apparent that more explicit communication must take place between protocol layers, and across network components. The co-ordinated delivery of interactive data requires that there not be conflict or disagreement on network state in the protocol stack. Only with a common interface, set of agreed communication paths, and organised information flow, can this be achieved.

Bibliography

[1] B. Ahlgren, M. Bjor̈kman, and B. Melander. Network Probing Using Packet Trains. Submitted to Global Internet '99, March 1999.

[2] M. Allman, C. Hayes, and S. Ostermann. An Evaluation of TCP with Larger Initial Windows. *ACM Computer Communication Review*, 28(3), July 1998.

[3] M. Allman and V. Paxson. On Estimating End-to-End Network Path Properties. In *ACM SIGCOMM '99, Cambridge, MA, USA*, September 1999.

[4] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. Internet RFC 2581, April 1999.

[5] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving Simulation for Network Research. Technical Report 99-702, University of Southern California, March 1999.

[6] A. Bestavros and G. Kim. TCP Boston - A Fragmentation-tolerant TCP Protocol for ATM Networks. In *Proceedings of IEEE INFOCOM'97*, 1997. http://cs-www.bu.edu/techreports/96-014-tcp-boston.ps.Z.

[7] T. Bonald. Comparison of TCP Reno and TCP Vegas via Fluid Approximation. Workshop on the Modelling of TCP, ENS, Paris, 1998.

[8] R. Braden. Requirements for Internet Hosts – Communication Layers. Internet RFC 1122, 1989.

[9] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. RFC 2205, Resource ReSerVation Protocol (RSVP) . Internet RFC 2205, 1997.

[10] L. Brakmo and L. Peterson. TCP Vegas: End to End Congestion Avoidance on a Global Internet. *IEEE Journal on Selected Areas in Communication*, 13(8):1465–1480, October 1995.

[11] L.S. Brakmo, S. W. O'Malley, and L.L. Peterson. TCP Vegas: New Techniques for Congestion Detection and Avoidance. In *Proceedings of SIGCOMM '94*, Aug 1994. ftp://ftp.cs.arizona.edu/xkernel/Papers/vegas.ps.

[12] R.L. Carter and M.E. Crovella. Dynamic Server Selection using Bandwidth Probing in Wide-Area Networks. March 1996.

[13] D. Comer and J. Lin. TCP Buffering and Performance Over an ATM Network. *Internetworking: Research and Experience*, 1995.

[14] R. Gallager D. Bertsekas. *Data Networks*. Prentice Hall, 1992.

[15] K. Djemame and M. Kara. Agent-Based Rate Coordination Between TCP and ABR Congestion Control Algorithms. *To appear in the Journal of Computer Communications - Special Issue on Performance Evaluation of Telecommunication Systems: Models, Issues and Applications.*

[16] R. Engel, D. Kandlur, A. Mehra, and D. Saha. Exploring the Performance Impact of QoS Support in TCP/IP Protocol Stacks. In *IEEE Infocom*, March 1998.

[17] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616, Hypertext Transfer Protocol – HTTP/1.1. Internet RFC 2616, 1999.

[18] S. Floyd. TCP and Explicit Congestion Notification. *ACM Computer Communication Review*, 24(5):8–23, Oct 1994.

[19] S. Floyd and V. Jacobson. Random Early Detection Gateways for Congestion Avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[20] J.C. Hoe. Improving the Start-up Behavior of a Congestion Control Scheme for TCP. In *Proceedings of the ACM SIGCOMM '96*, pages 270–280, August 1996.

[21] V. Jacobson. Congestion Avoidance and Control. *ACM Computer Communication Review*, 18:314–329, Aug 1988. Proceedings of Sigcomm'88 Symposium, Stanford, CA.

[22] V. Jacobson. Modified TCP Congestion Avoidance Algorithm. end2endinterest mailing list, April 1990.

[23] R. Jain, D. Chiu, and W. Hawe. A Quantitative Measure Of Fairness And Discrimination For Resource Allocation In Shared Computer Systems. Technical Report TR-301, DEC, September 1984. http://www.cis.ohio-state.edu/~jain/papers/fairness.htm.

[24] F. Kelly. Charging and rate control for elastic traffic. *European Transactions on Telecommunications*, 8:33–37, 1997.

[25] S. Keshav. REAL: A Network Simulator. Technical Report 88/472, Department of Computer Science, UC Berkeley, 1988.

[26] S. Keshav. A Control-Theoretic Approach to Flow Control. In *Proceedings of ACM SIGCOMM 1991*, 1991.

[27] S. Keshav. *Congestion Control in Computer Networks.* PhD thesis, Department of EECS, UC Berkeley, 1991.

[28] S. Keshav. Flow Control in High-Speed Networks with Long Delays. In *Proceedings of INET 1992*, June 1992.

[29] S. Keshav. *An Engineering Approach to Computer Networking*. Addison Wesley, 1997.

[30] S. Keshav, A. Agrawala, and S. Singh. Design and Analysis of a Flow Control Algorithm for a Network of Rate Allocating Servers. *Protocols for High Speed Networks II*, April 1991. Published on IFIP Press.

[31] S. Keshav and A. Banerjea. Queueing Delays in Rate-Controlled Networks. In *Proceedings Infocom 1993*, March 1993.

[32] J.Roberts L. Massouile. Bandwidth sharing: objectives and algorithms. CNET-France Telecom, 1998.

[33] M. Mathis, J. Madhavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. Internet RFC 2018, Oct 1996.
ftp://ftp.ee.lbl.gov/papers/rfc2018.ps.

[34] J. Mo, R. La, V Anantharam, and J Warlrand. Analysis and Comparison of TCP Reno and Vegas. In *INFOCOM 99*, 1999.

[35] J. Nagle. Congestion Control in IP/TCP Internetworks. Internet RFC 896, 1984.

[36] K. Nichols, S. Blake, F. Baker, and D. Black. RFC 2474, Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. Internet RFC 2474, 1998.

[37] C. Partridge, S. Floyd, and M. Allman. Increasing TCP's Initial Window. Internal Draft, Internet Engineering Task Force, Sep 1998.
ftp://ftp.isi.edu/in-notes/rfc2414.txt.

[38] V. Paxson. End-to-end internet packet dynamics. *ACM SIGCOMM Computer Communication Review*, 27(4):139–152, October 1997.

[39] K. Poduri and K. Nichols. Simulation Studies of Increased Initial TCP Window Size. Informational Internet RFC 2415, 1998.

[40] J. Postel. RFC 768, User Datagram Protocol. Internet RFC 768, 1980.

[41] J. Postel. RFC 793, Transmission Control Protocol. Internet RFC 793, 1981.

[42] K. Ramakrishnan and R. Jain. A Binary Feedback Scheme for Congestion Avoidance in Computer Networks. *ACM Transactions on Computer Systems*, 8(2):158–181, 1990.

[43] A. Romanow and S. Floyd. Dynamics of TCP Traffic over ATM Networks. *IEEE Journal on Selected Areas in Communications*, 13(4):633–641, May 1995.

[44] S. Floyd et al. Tcp friendly web site. http://www.psc.edu/networking/tcp_friendly.html.

[45] H. Schulzrinne, S. Casner, and V. Jacobson R. Frederick. A Transport Protocol for Real-Time Applications. Internet RFC 1889, 1996.

[46] H. Schulzrinne, A. Rao, and R. Lanphier. Real Time Streaming Protocol (RTSP). Internet RFC 2326, 1998.

[47] T. Shepard and C. Partridge. When TCP Starts Up With Four Packets Into Only Three Buffers. Informational Internet RFC 2416, 1998.

[48] W. Stevens. TCP Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery Algorithms. Internet RFC 2001, 1997.

[49] R. Wade, M. Kara, and P.M. Dew. Proposed Modifications to TCP Congestion Control for High Bandwidth and Local Area Networks. In *Proceedings of the 6th IEEE Conference on Telecommunications (ICT'98)*, July 1998.

[50] R. Wade, M. Kara, and P.M. Dew. Modeling and Simulation of STTP, a Proactive Transport Protocol. In *Proceedings of the IEEE International Conference on Networking (ICON 2000)*, pages 485–486, September 2000.

[51] R. Wade, M. Kara, and P.M. Dew. Study of a Transport Protocol Employing Bottleneck Probing and Token Bucket Flow Control. In *Proceedings of the IEEE International Symposium on Computer Communications (ISCC 2000)*, pages 225–229, July 2000.

[52] Z. Wang and J. Crowcroft. A New Congestion Control Scheme: Slow Start and Search (Tri-S). *ACM Computer Communication Review*, 21:32–43, 1991.

[53] Z. Wang and J. Crowcroft. Eliminating Periodic Packet Losses in the 4.3-Tahoe BSD TCP Congestion Control Algorithm. *ACM Computer Communication Review*, 22:9–16, April 1992.

Table 1: Table of Results for Experiment 2

Num Sources	Protocol	bytes tx	bytes drop
10	Reno	226345000	1264000
10	Vegas	225025000	0
10	STTP	225064800	0
20	Reno	230285000	5190000
20	Vegas	225370000	283000
20	STTP	225520600	435000
30	Reno	234791000	9696000
30	Vegas	230131000	5043000
30	STTP	225997400	904080
40	Reno	239531000	14436000
40	Vegas	236000000	10911000
40	STTP	226253200	1161160
50	Reno	241044000	15954000
50	Vegas	241185000	16096000
50	STTP	227099000	2005120
60	Reno	243972000	18880000
60	Vegas	244975000	19888000
60	STTP	226860800	1767680
70	Reno	247247000	22165000
70	Vegas	248030000	22948000
70	STTP	227146600	2054360
80	Reno	250221000	25126000
80	Vegas	250684000	25595000
80	STTP	227614400	2518360
90	Reno	253429000	28334000
90	Vegas	253156000	28079000
90	STTP	227507200	2412560
100	Reno	255582000	30487000
100	Vegas	255914000	30825000
100	STTP	227681000	2585840
110	Reno	257234000	32142000
110	Vegas	258223000	33135000
110	STTP	227537800	2442600
120	Reno	258980000	33890000
120	Vegas	259093000	34004000
120	STTP	227872600	2777280
130	Reno	261137000	36042000
130	Vegas	262281000	37192000
130	STTP	228831400	3737800
140	Reno	262140000	37046000
140	Vegas	262223000	37138000
140	STTP	228737200	3641840
150	Reno	262968000	37873000
150	Vegas	264653000	39564000
150	STTP	228412000	3315920
160	Reno	264286000	39195000
160	Vegas	266033000	40946000
160	STTP	229956800	4861520
170	Reno	264828000	39733000
170	Vegas	267471000	42382000
170	STTP	229206600	4111640
180	Reno	265320000	40225000
180	Vegas	268390000	43302000
180	STTP	229552400	4456400
190	Reno	265871000	40776000
190	Vegas	269193000	44105000
190	STTP	228915200	3819680
200	Reno	265593000	40501000
200	Vegas	268748000	43660000
200	STTP	229738000	4641920

Table 2: Table of Results for Experiment 4

Num Sources	Protocol	bytes tx	bytes drop
10	Reno	57884000	1608000
10	Vegas	56302000	3000
10	STTP	56895800	598000
20	Reno	60352000	4063000
20	Vegas	58348000	2044000
20	STTP	56770600	469000
30	Reno	61939000	5642000
30	Vegas	61468000	5165000
30	STTP	56989400	688000
40	Reno	63661000	7356000
40	Vegas	63087000	6785000
40	STTP	57251200	949120
50	Reno	65407000	9102000
50	Vegas	64992000	8687000
50	STTP	57503000	1200040
60	Reno	67252000	10962000
60	Vegas	66076000	9772000
60	STTP	57719800	1416160
70	Reno	68611000	12306000
70	Vegas	67666000	11363000
70	STTP	57931600	1629160
80	Reno	69429000	13128000
80	Vegas	68590000	12287000
80	STTP	58255400	1952040
90	Reno	70152000	13855000
90	Vegas	69982000	13680000
90	STTP	58596200	2291000
100	Reno	70927000	14623000
100	Vegas	70463000	14159000
100	STTP	58798000	2494040