Modelling and Simulation of STTP, a Proactive Transport Protocol

R.Wade, M.Kara, P.M.Dew {rik, mourad, dew}@scs.leeds.ac.uk
ATM-Multimedia Group, School of Computer Studies,
University of Leeds, Leeds, LS2 9JT, UK

Abstract— **In this paper, we present an experimental transport protocol which employs alternative methods of startup and congestion avoidance to those commonly used in implementations of TCP. Through modelling and simulation, we demonstrate how a packet-pair probe may be used to replace traditional slow-start methods and how proactive congestion avoidance algorithms provide smoother traffic flow with lower levels of packet loss and retransmission. We also show how a token bucket may be used to allow bounded burstiness for an Internet connection in order to facilitate an application's Quality of Service.**

I. Overview

STTP is an experimental transport protocol designed to operate primarily over traditional IP-based Internetworks. It is a Shaped (through use of leaky and token bucket traffic shapers), Token-based (packet acknowledgements are treated as tokens for use in its internal shaping algorithms), Transport Protocol (STTP). STTP is designed as an alternative for the reliable transport transport layer currently occupied by TCP, the de facto Internet standard.

STTP is a Quality of Service (QoS) oriented transport layer designed to improve on the existing Transmission Control Protocol (TCP) in several key areas:

- connection startup and recovery at least as efficient as TCP;
- direct support for QoS through traffic shaping;
- proactive congestion avoidance, and
- flexible, modular, expandable design.

High bandwidth, multimedia and realtime connections have very different requirements to that of standard *best effort* traffic. In particular, they are much more sensitive to packet loss and variations in bandwidth [1]. In order to address these issues, several attempts have been made to ameliorate existing protocols [2], [3], [4] while maintaining compatibility with the existing Internet standards [5].

Despite many modifications, TCP's congestion avoidance algorithms are not suited to modern network and traffic requirements [1]; In particular those with dedicated, highly reliable connections or in environments where flow control is needed on a per-connection basis. In these situations, TCP is unable to provide the necessary functionality to the application layer due to its reliance on window-based startup and congestion avoidance. Without significant re-engineering of the core algorithms, this sort of application support is not possible within the existing framework.

In response to these deficiencies, we have attempted tuning of TCP's congestion avoidance algorithms [6], but conclude that the basic design of the protocol is ill-suited to modern applications and network dynamics [7]. Therefore, an alternative protocol has been designed which provides all the features of TCP, but with a more flexible and powerful core mechanism.

The second section of this report provides an overview of STTP and its features, section three provides detailed, explanatory models of the protocol and section four gives comparative simulation results using both STTP and TCP Reno. Our conclusions are presented in section five and section six outlines plans for future work.

II. STTP: A Shaped, Token-based Transport Protocol

STTP is designed to provide an alternative transport layer to networked applications such as FTP, the World Wide Web (WWW) and Telnet. In order to resolve some issues with TCP's handling of multimedia traffic [1], and modern Internet applications' reliance on Variable Bit Rate (VBR) and bursty traffic streams [7], TCP's window-based flow control has been replaced with a token bucket mechanism. This allows bounded burstiness for a connection while maintaining a mean transmission rate over a given period of time (described in detail by Tanenbaum in [8]).

For connection initialisation we have deployed a packet-pair probe, which is used to initialise the token bucket's variables with enough tokens to utilise a link's currently available bandwidth. This technique was first described by S.Keshav in [9] but has been further examined by Carter and Crovella in [10]. A packet-pair probe involves transmitting two packets back-to-back to a remote host. When received, they are returned to the sender who analyses

the inter-arrival gap. This gap can be used to calculate the available bandwidth on a connection's bottleneck link. Therefore, we obtain the maximum allowable transfer rate for a new connection.

Once established, STTP uses a proactive congestion avoidance algorithm, which is sensitive to fluctuation in a connection's Round Trip Time (RTT). A similar method is used by TCP Vegas. TCP Vegas's [3] congestion control consists of a bounded *window* within which a connection's Round Trip Time (RTT) is allowed to vary. If the actual RTT falls outside this *predicted*, then the transmission rate is adjusted accordingly. For example if the RTT increases, this indicates that more traffic has entered a connection's path and both router queues and network connections are becoming congested. If this increase is within a connection's window of tolerance, no change is made to the transmission rate. However, should the RTT increase noticeably, then STTP may reduce its transmission rate so as to avoid packet loss. Conversely, if a connection's RTT decreases, STTP may increase its traffic flow to take advantage of the newly available bandwidth. This approach is shown to be very effective for avoiding packet loss and data retransmission (see section 4).

Each packet transmitted contains a sequentially numbered token which is returned by the receiver in order to verify safe delivery. Should a token not be returned within a given time then retransmission will occur. In our experiments, we used standard TCP Reno timeout values and packet retransmission algorithms. This mechanism is depicted in figure 1.

A. Related Work

The original specification for TCP's slow start and congestion avoidance algorithms can be found in [11]. Since that time, the late 1990's have seen massive growth in the use of Internet-based computer applications such as the World Wide Web and a shift in network traffic types and requirements. While Jacobson's recommendations were well suited for applications such as FTP, bursty, multimedia traffic has created the need for next-generation protocols to be examined. Revision of the TCP protocol has been carried out by W.Stevens in [2] in order to address issues with its congestion avoidance algorithms in heavily congested networks. Further examination was performed by Brakmo et. al in [3] with their work on TCP Vegas. Their recommendations included a proactive congestion avoidance algorithm combined with a more cautious startup mechanism. S.Keshav, however, previously designed a protocol based on the packet-pair probe [9] which is examined closely in his thesis, [12]. This probe mechanism allows a node to discover the currently available

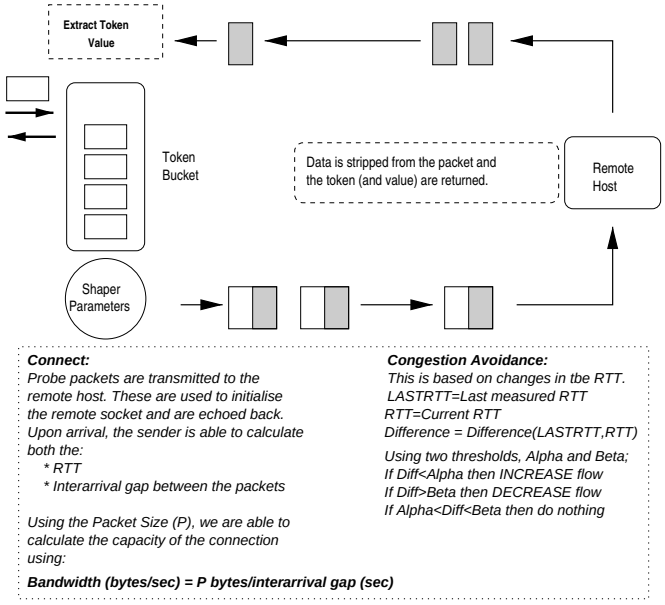


Fig. 1. STTP Block Diagram

bandwidth on a given connection in roughly one round-trip time and is utilised in our work with STTP.

More recently, work has been conducted by Carter and Crovella [10] and Allman and Paxson [13], who have examined the application of packet-pair techniques in layer 4 transport protocols.

III. Model Validation

In order to validate the model laid out in [14] (section 3) we performed an incremental implementation of STTP in the 'ns' network simulator [15]. By adopting this method, we were able to test and validate each element of the protocol in isolation while building towards a fully functional simulation model.

A. Packet Pair Probe

A simple Ping agent was developed for ns which could be bound to both transmitting and receiving nodes. When receiving a packet, it checks to see whether it has already been returned by another agent. If not it is transmitted without delay back to the transmitting node. If the packet has already passed through a Ping agent, a reading of its round trip time (RTT) is taken.

By transmitting two such packets back-to-back across a test network, we are able to measure their respective RTT's and hence, their inter-arrival gap when returned. From Keshav's work on packet pair probes [9], we know that $Bandwidth(bytes/sec) = \frac{PacketSize(bytes)}{InterArrivalGap(seconds)}$.

The Inter Arrival Gap (IAG) can be further described as the delay incurred when probe packets are processed

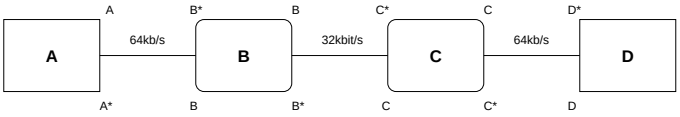


Fig. 2. A sample network

by the *entry edge* of a connection's bottleneck link. For example, consider the network in figure 2.

The bottleneck connection in this case is between routers B and C. The connection here is 32kb/s (kilo-bits per second) as opposed to 64kb/s for link A,B and C,D. This means that packets entering router B or C may be delayed while earlier traffic is transmitted over the bottleneck link. Needless to say that if the traffic transmitted between A,B or D,C exceeds the bottleneck capacity, the queue at the entry edge router will increase it reaches capacity and data is lost.

If data enters router B at speed of 64kb/s, the fastest it can be forwarded over link B,C is 32kb/s. If we term the arrival (RX) side of this router to be B and the departure (TX) size to be $B*$, we can conclude that $C = B*$ and that $B* <= B = A*$. If 64bytes of data arrives at B, it can only be forwarded at a rate of 32kb/s, taking 16ms. This will introduce a delay of up to 16ms for any packets arriving within this period. We can therefore calculate the bottleneck delay to be $\frac{PSize(bits)}{X-X*}$, where X is a router at the leading edge of the bottleneck link. As both probe packets are transmitted back-to-back from A, the second packet will be queued while the first is forwarded onto the link from $B*$. As they will be back-to-back upon entering this router, the maximum delay will be incurred (16ms).

In order to validate this model, the above topology was entered into the 'ns' simulator and our packet-pair startup probe run over it. Each 64Kb link had a latency of 5ms and the 32Kb, 10ms. The packet size was 64bytes. Packets 1 and 2 were transmitted back-to-back from A. They returned at 104ms and 120ms respectively with an inter-arrival gap of 16ms. Using the formula $Bandwidth(bytes/sec) = \frac{PacketSize(bytes)}{InterArrivalGap(seconds)}$, we obtain $Bandwidth(bytes/sec) = \frac{64}{0.016}$ which gives us 4000bytes/second, or 32kb/s.

This simulation was run with a range of bottleneck values from 1kb/s through to 155Mbps. Furthermore, the bottleneck was shifted from the mid-point to start and end links. The packet-pair probe was reported within an accuracy of 2% in all tests. DropTail queueing was used on both routers.

B. Token Bucket Model

The traditional model for a token bucket is that of a σ, ρ mechanism where σ is the maximum depth of the bucket and ρ the feed rate of tokens. This means that the bucket will fill at rate ρ until it reaches σ, after which time tokens will be discarded.

The source is permitted to send data at the maximum rate while there are tokens in the bucket (i.e. it is in credit). Tokens are normally assigned a byte-value, which means that if a bucket contains 10*1024 byte tokens, the source can send up to 10KB of data in a single burst. However, this does not take into account tokens which arrive during the burst period.

Given a bucket capacity σ, a token arrival rate ρ and a burst of length S, it is evident that $\sigma + (\rho * S)$ is the maximum burst size. If the maximum output rate from our source is M (bytes/sec), then $M * S$ is the number of bytes in a burst of length S. Tannenbaum shows that we can illustrate the maximum length burst from a given bucket with $S = \sigma/(M - \rho)$.

STTP uses a variation on the token bucket model in order to prevent blind injection of packets into a network. While a fixed-rate flow of tokens could be scaled according to available bandwidth on a connection, it appears more sensible to adopt the approach of a self-clocking protocol such as TCP, which uses acknowledgement (ACK) packets to indicate successful data delivery.

Therefore, this experimental version of STTP uses the successful receipt of an ACK to trigger the release of further tokens into the bucket. While not as regular as a traditional ρ flow, it is anticipated that ACK arrivals will be based around a mean value which can be modeled as a Poisson process [16].

Given the above, it is possible to model the arrival of packets over a given time period with the following algorithm. It takes λ and t as arguments. λ is the mean arrival rate expected, and t is the number of iterations to perform (hence the number of time steps being processed).

This provides us with a function to describe the arrival of ACK packets to the token bucket. ACKs are normally received individually and independently of other packets, hence can be modeled in this way. Given the way in which the token bucket functions, a source is permitted to send data if there are sufficient tokens available. Therefore, if a greedy source is used, we would expect to see a transmission profile fitting that of the arrival process. In order to test this theory, the above algorithm was implemented in C. When run, it outputs the number of packets (ACKS) received in each time period. The network shown in figure 2 was used to validate this model.

TABLE I
VALIDATION OF THE POISSON AND SIMULATION MODELS

Network Configuration	Duration (seconds)	Poisson Events (tokens generated)	Simulation Events Packets TX [ACKs]
64Kb links with 1ms latency	10	73	75 [68]
64Kb links with 1ms latency	50	417	421 [388]
128Kb links with 0.5ms latency	10	168	171 [158]
128Kb links with 0.5ms latency	50	821	819 [798]
2Mb 1ms, 1Mb 2ms, 2Mb 1ms	20	2518	2500 [2458]
128Kb 1ms, 1Mb 0.5ms, 2Mb 0.1ms	40	655	658 [640]

In order to generate the Poisson (ACK) events in table I, the mean number of events was declared as the capacity of the bottleneck link divided by the packet size. This gives the number of packets per second that can be processed by the bottleneck, and also, the capacity to which the STTP's token bucket is initialised. Using this value as the mean for our Poisson process, and the length of the simulation (in seconds) for the number of iterations, we implemented a small C program to generate Poisson arrival values.

B.1 Greedy Source Model

We are now able to model the arrival of ACK packets (given in square brackets in table I). Similarly, if a greedy FTP source is used in our simulation (for example), we also noted that the number of packet transmissions closely matched that of the Poisson model. This is due to the FTP application always having data ready to send and therefore using tokens as soon as they arrive in the bucket. We will now expand on this model to address the area of bursty applications such as an interactive conferencing session, or the World Wide Web (HTTP).

With STTP running at the transport layer, an application is able to transmit up to σ packets at any one time. Provided there are sufficient tokens in the bucket, these will be transmitted and queued at the data-link layer pending space on the physical connection. This means that should an application need to send bursts of data, it is able to do so provided sufficient ACKs have been received from earlier transmissions to fill the token bucket.

Combined with a packet-pair probe, which can accurately estimate the currently available bandwidth on a given connection, we can initialise the token bucket in a manner which will allow reliable burstiness without going through slow start (as with TCP). The advantage of this is that short bursts of data, as often seen in HTTP transfers, can be carried with a much shorted connection time.

In order to illustrate this behaviour, we consider a simple example where an application wishes to send just a single burst of data. A three-node topology with 64kb/s links (1ms latency) was implemented in 'ns' with node 0 being the source and node 2 the destination. An FTP application was then asked to generate 8 packets of data (from node 0 to node 2) and we monitored its behaviour. The network topology was similar to that in figure 2, but consisted of only three nodes. A sender, a router and a receiver.

STTP's token bucket was able to transmit seven of the eight packets at t=0.029(seconds) after having initialised itself with a packet-pair probe. The eighth packet was queued for subsequent transmission at t=0.668. The first acknowledgement was received at t=0.281, the last at t=1.156. TCP Reno, on the other hand, was able to start transmitting data at t=0.0 and received its first acknowledgement at t=0.264. However, its last acknowledgement from the traffic burst did not arrive until t=1.292. In this scenario, STTP offered a 10% performance increase over TCP Reno due to its fast startup and token bucket flow control. TCP Reno's burst speed was affected by it having to go through slow start and open its congestion window.

B.2 Bursty Source Model

Given an on period of length Alpha and an off period of length Beta, we are able to model STTP's capacity for dealing with on-off, bursty traffic sources such as interactive network applications. The following assumes that an on period (Alpha) is preceded by an off period (Beta). In an on period, the sender is able to transmit ((ρ*(Alpha+Beta))+num_tokens), where num_tokens is the number of tokens already present in the bucket at the beginning of period Beta and ρ is the feed rate of tokens (previously modeled as a Poisson arrival).

In order to demonstrate this model in practice, a simple 3-node network toplogy was implemented in 'ns' (nodes 0,1 and 2). Each node was connected with 64kb/s links and 1ms latency. Droptail queueing was used at node 1, the router. An FTP source was located at node 0 and run over STTP (over IP). Traffic was directed to node 2, a modified

TCPSink. Two events were scheduled, at t=0.0, the FTP source was requested to transmit 8 packets. Then, at t=2.0, to send indefinitely. The simulation terminates at t=50.

STTP's token bucket was successfully initialised to 8 packets by the packet-pair probe. This credit was immediately used by the FTP source to send its data. The Poisson arrival model for tokens indicates that up to 10 ACK's should be received in the first second. However, as in our previous experiment, the final packet of data was not transmitted until t=0.668, some time after the initial burst. Therefore all ACK's from the transmitted packets were received by t=1.156. The token bucket is then ready for the scheduled burst at t=2.0.

By the end of the experiment, t=50, a comparison between STTP and TCP Reno shows that both transmitted the same number of packets (391). This implies that in addition to short-term benefits, STTP performs at least as well as TCP Reno in longer transfers on uncongested networks.

B.3 Congestion Avoidance Model

STTP bases its congestion avoidance algorithms on those of TCP Vegas [3], which are essentially sensitive to variations in Round Trip Time (RTT). We take measurements of the time it takes for a data packet to be transmitted and the relevant acknowledgement to be received. If a network is becoming more congested, then queue lengths will increase at routers along a given connection. As a result, the RTT will increase. TCP Vegas actually measures the difference between expected and actual throughput over a given time period. STTP takes measurements of a connection's RTT and, according to certain boundaries, will adjust its flow of data accordingly. Simulation experiments have shown that variations of +/-5% should be considered significant. Values smaller than this encouraged very rapid fluctuation, and those of 10% or greater meant that many instances of congestion were missed, resulting in loss of data over time.

In [6], we discussed the use of weighted averages which can be employed to prevent a protocol reacting too quickly to what may be an anomalous event. An example of this would be temporary loss of input from one data connection, resulting in a rapid drop in router utilisation. STTP also uses this mechanism when reacting to variations in RTT. As with our TCP congestion avoidance modifications, we use a 0.5, 0.5 weighted average to recalculate σ and ρ. This issue is discussed further in section 4's discussion.

Should a new RTT be greater than +/-5% of the previous measurement, then σ, ρ and the current number of tokens in the bucket are all scaled accordingly. For example, $\sigma = ((0.9 * \sigma) + (0.1 * (\sigma * (lastRTT/newRTT))))$ is the scaling formula for the size of the token bucket. Replacing σ with ρ will scale the feed rate. All of the bucket's attributes are scaled with variations in a connection's RTT. This means that STTP is able to dynamically adjust its quality of service to match the currently available resources.

We would therefore expect an active STTP connection to adapt its data flow, smoothly yet quickly, to the currently available bandwidth. In order to demonstrate this mechanism, a network topology was implemented in 'ns' with two traffic sources. The first source, a constant bit-rate source was started at t=0 to send (1000 byte) packets at 0.08 second intervals. The second source, an STTP source, was timed to start at t=1.0. The two sources were connected via 128kb/s (10ms latency) links to a 128kb/s (10ms latency) backbone. Each source had its own dedicated target node, connected to the backbone via a 128kb/s, 10ms link.

As STTP began its transmission at t=1.0, it probed the bottleneck bandwidth to be its local link of 64kb/s. Its first data transmissions encountered slight delay due to queue build-up from CBR cross-traffic, a RTT of 0.255 seconds was measured, compared with 0.078 seconds during the packet-pair probe. Therefore, STTP scaled σ, ρ and the number of available tokens by $x = ((0.9 * x) + (0.1 * (x * (lastRTT/newRTT))))$, where x is the appropriate attribute. Taking the case of σ, $lastRTT/newRTT = 0.305$ and σ's previous value was 16. (0.9*16)+(0.1*(16*0.305))=14.888 (calculated to 3 decimal places for clarity), which yields a new token bucket size of 15. This result was confirmed by our 'ns' model of STTP in the scenario described above.

The CBR source was timed to cease transmission at t=2.5. Being a non-responsive source, this freed the fixed amount of bandwidth which had been previously required. As the CBR source's final packets filtered through the network's router queue, the RTT for STTP packets gradually increased until, at t=2.8, the first ACK with a greatly reduced RTT arrived. The new RTT was 0.938, compared with 1.630 for the last packet acknowledgement. STTP reacted to this drop by increasing its data flow in line with our model.

In this section, we have shown how our token bucket and packet-pair probe models have been implemented in the 'ns' simulator and how its performance on simulated network topologies falls within our expectations.

IV. STTP Performance

TCP Reno, the most widely used variant of TCP on the Internet at this time, is often criticised for its congestion

avoidance and startup methods. Reno can only detect congestion by essentially losing data. It therefore tends to be relatively aggressive in its use of available bandwidth. Furthermore, its slow start, exponential window increase, algorithm is a common cause for packet loss early on in a connection [3].

In our performance testing of STTP, we encountered both positive and negative aspects of the chosen algorithms. When assessing performance, we consider raw throughput, packet loss (a function of which is goodput), fairness and stability. Raw throughput and packet loss are easily obtained statistics from a simulator. For a given connection, the number of packets transmitted minus the number dropped, yields its goodput. This metric is becoming more important with heavily congested backbones and Internet services. In many cases it is preferable to have a more consistent, but lower, throughput than a bursty but lossy connection.

We therefore consider two further derived metrics, fairness and stability. Much modelling and discussion of fairness has been performed by research groups such as [17]. The definition that we consider in this work is that of max-min fairness [18]. Stability, on the other hand, implies that a protocol should ideally change state smoothly and not produce packet bursts which are detrimental to the network's current state. For example, to enter an already congested network with a large burst of data would do little to improve its state. It would be much more preferable to introduce packets more slowly and cause competing connections to relinquish some of their bandwidth without packet loss.

A. Simulations

The network topology used for the performance testing of STTP can be seen in figure 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 [19] simulator.

By varying the number of sources and the link capacity, we are able to create a wide variety of scenarios under which to test STTP, TCP Reno and TCP Vegas. Our reference protocol is TCP Reno, as a large amount of modelling and simulation has been conducted in the past, which provides a clear understanding of its positive and negative attributes. The majority of TCP implementations currently in use are based on BSD Reno with some beginning to use Vegas-like pro-active congestion avoidance.

In all experiments, the links from nodes *t*n to router *a* are 10Mb/s capacity, 1.0ms latency. Queue capacity at all routers is capable of holding 128 packets (packets are

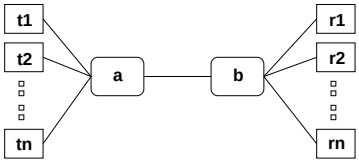


Fig. 3. Performance Simulation Topology

1000bytes, so 128KB in total). The main backbone link consists of a 2Mb/s capacity, 4.0ms latency connection. The final link from router *b* to *r*n is of 10Mb/s capacity and 1.0ms latency. Each transmitting node has a greedy FTP source agent attached to it, which sends data to the corresponding *t* node, a sink which accepts TCP socket connections (compatible with STTP).

Experiment 1, the raw data for which is depicted in table III, Appendix A, uses the above network topology. Simulations were run with between 10 and 200 sources, incremented by 10 sources with each iteration and starting at 4 second intervals. Each scenario was run with TCP Reno, TCP Vegas and STTP. The data in table III 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.

Figure 5 depicts the raw data for dropped packets from table III, while figure 4 graphs the acutal *goodput* (transmitted packets - dropped packets) for each simulation.

B. Discussion

In table III, 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 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.

Here, STTP is shown to have goodput comparable with that of TCP Reno. 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 second graph, figure 5. 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.

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.

Fig. 4. Experiment 1: TCP Reno, TCP Vegas and STTP Successfully Received Packets

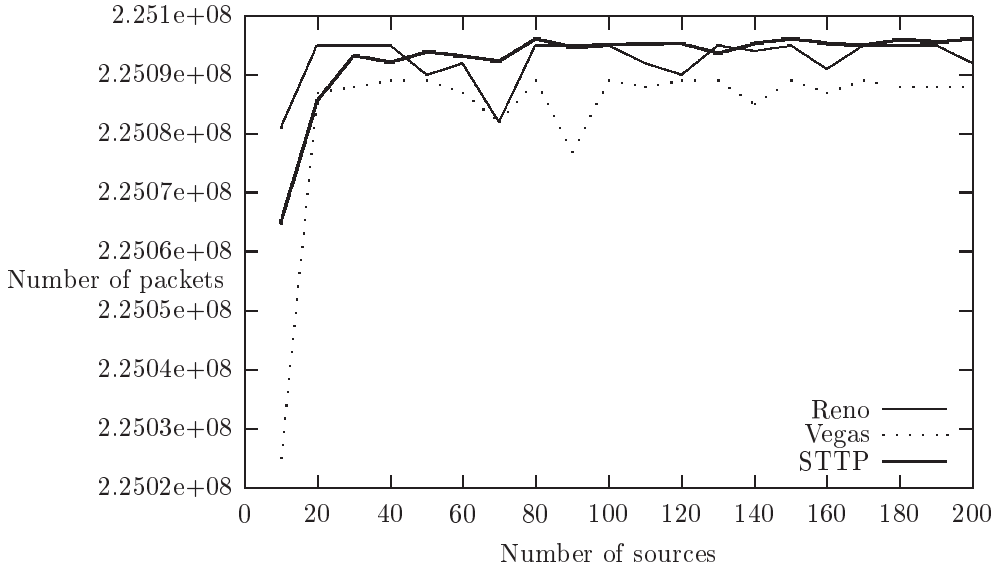
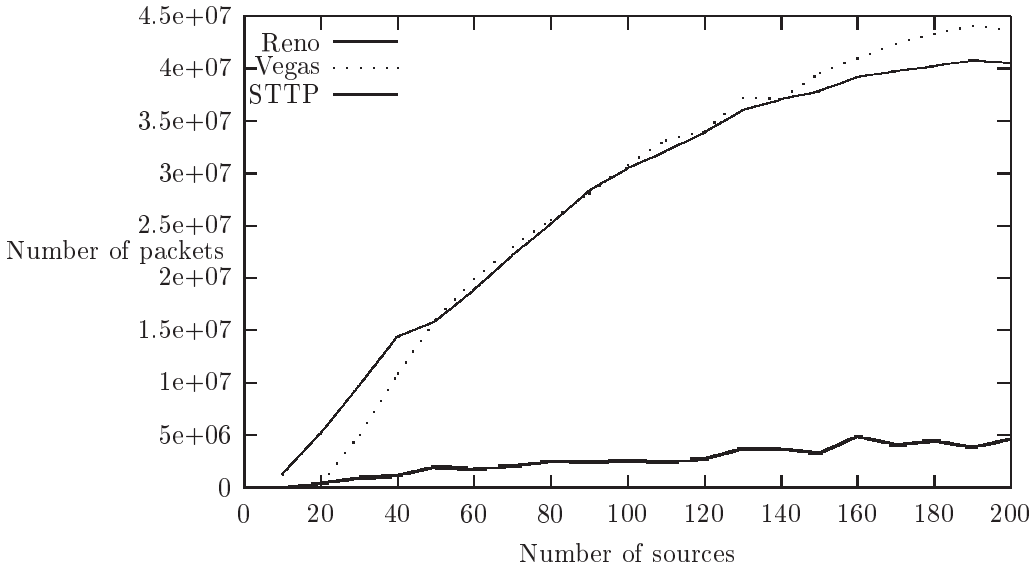


Fig. 5. Experiment 1: TCP Reno, TCP Vegas and STTP Total Packet Loss



However, it is also important to share these resources efficiently between competing connections.

B.1 Fairness

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. In all cases, STTP’s packet loss is less than 50% of that of TCP Reno.

However, the disadvantage of STTP’s congestion avoid-

ance 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 II shows the results of further simulations conducted on a similar topology to that used in previous experiments. 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 II 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 II 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 [20], 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 II, 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, unfair.

Traditional max-min fairness [18] states that given a set of limited network resources, 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 II 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 II 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 experiments, 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.

V. Conclusions

A. Summary

In this paper, we have presented our work with a prototype transport protocol and through modelling and simulation, shown it to offer several distinct advantages over the current de facto standard, TCP Reno. STTP exhibits far lower packet loss than TCP Reno, yet is able to utilise available bandwidth on a link in a shorter period of time due to its fast startup algorithm. It will share bandwidth fairly among competing streams and lose fewer packets at bottleneck links. Such congestion avoidance is highly beneficial when considering interactive or realtime Internet applications. While its delay-sensitive congestion avoidance algorithm is not as fast as that of TCP Reno to adjust to new traffic, it does provide stable, fair allocation of bandwidth amongst competing TCP friendly streams [21] [22].

B. Future Work

The version of STTP used for this paper, and in [6], employed a simplified token bucket model, based on the receipt of packet acknowledgements to trigger credit release (ρ). While this has some advantages over a more traditional timer-based token-bucket model, further work has shown significant performance improvements using the latter technique. A further advantage is that the bucket does not have to be initialised with a given number of to-

TABLE II
STTP FAIRNESS

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

kens in order to ”kick start” the packet flow. With a timer-based system, tokens will be produced at regular intervals despite the current network state. This could, however, be seen as a disadvantage and was an important factor in our initial decision to use ACK-based flow control.

Unfortunately, despite higher overall goodput, our timer-based model does incur higher packet loss although this is still lower than TCP Reno under the same conditions. The precise dynamics of STTP's congestion avoidance can, however, be directly manipulated with its history-weighted bucket and flow calculations. By increasing or decreasing the bias towards current or past events, the protocol can be altered from aggressive to passive in nature. An interesting branch of study would be to examine the feasibility of *adaptive* values for this component.

A further avenue of interest is that work conducted by Ahlgren et al [23], where the packet-pair technique is expanded to include chains of probe packets. Their results correspond with those of Carter and Crovella [10], and show stable estimations with trains of ten to fifteen packets. As the train increases in length, as does the accuracy of the bottleneck bandwidth estimation. This would also help prevent startup STTP streams from failing due to initial loss of probe data.

APPENDIX

Table III shows the raw data from Experiment 1.

TABLE III

EXPERIMENT 6

Num Sources	Protocol	bytes tx	bytes drop	mean bytes	high bytes	low bytes
10	Reno	226345000	1264000	22634500	36411000	6760000
10	Vegas	225025000	0	22502500	45062000	11283000
10	STTP	225064800	0	22506480	27922080	10692080
20	Reno	230285000	5190000	11514250	40050000	5863000
20	Vegas	225370000	283000	11268500	22905000	2200000
20	STTP	225520600	435000	11276030	31669080	1204080
30	Reno	234791000	9696000	7826367	40906000	4838000
30	Vegas	230131000	5043000	7671033	11817000	3058000
30	STTP	225997400	904080	7533247	27842080	325080
40	Reno	239531000	14436000	5988275	11055000	4952000
40	Vegas	236000000	10911000	5900000	8408000	2713000
40	STTP	226253200	1161160	5656330	21002080	80
50	Reno	241044000	15954000	4820880	14958000	3295000
50	Vegas	241185000	16096000	4823700	8836000	1913000
50	STTP	227099000	2005120	4541980	13745080	80
60	Reno	243972000	18880000	4066200	13686000	2814000
60	Vegas	244975000	19888000	4082917	8014000	1920000
60	STTP	226860800	1767680	3781013	16009080	80
70	Reno	247247000	22165000	3532100	13379000	2415000
70	Vegas	248030000	22948000	3543286	8459000	1609000
70	STTP	227146600	2054360	3244951	12894080	80
80	Reno	250221000	25126000	3127763	12733000	1960000
80	Vegas	250684000	25595000	3133550	5937000	1667000
80	STTP	227614400	2518360	2845180	8360080	80
90	Reno	253429000	28334000	2815878	8020000	1680000
90	Vegas	253156000	28079000	2812844	6700000	1012000
90	STTP	227507200	2412560	2527858	9375080	80
100	Reno	255582000	30487000	2555820	7878000	1116000
100	Vegas	255914000	30825000	2559140	6435000	706000
100	STTP	227681000	2585840	2276810	7712080	80
110	Reno	257234000	32142000	2338491	13030000	715000
110	Vegas	258223000	33135000	2347482	6569000	983000
110	STTP	227537800	2442600	2068525	11078080	80
120	Reno	258980000	33890000	2158167	9255000	7000
120	Vegas	259093000	34004000	2159108	6727000	349000
120	STTP	227872600	2777280	1898938	6643080	80
130	Reno	261137000	36042000	2008746	13521000	574000
130	Vegas	262281000	37192000	2017546	5512000	575000
130	STTP	228831400	3737800	1760242	6704080	80
140	Reno	262140000	37046000	1872429	12890000	6000
140	Vegas	262223000	37138000	1873021	5944000	332000
140	STTP	228737200	3641840	1633837	6298080	80
150	Reno	262968000	37873000	1753120	8186000	9000
150	Vegas	264653000	39564000	1764353	5847000	314000
150	STTP	228412000	3315920	1522747	6817080	80
160	Reno	264286000	39195000	1651788	7189000	6000
160	Vegas	266033000	40946000	1662706	5753000	244000
160	STTP	229956800	4861520	1437230	5649080	80
170	Reno	264828000	39733000	1557812	7671000	9000
170	Vegas	267471000	42382000	1573359	5492000	272000
170	STTP	229206600	4111640	1348274	6282080	80
180	Reno	265320000	40225000	1474000	12280000	6000
180	Vegas	268390000	43302000	1491056	5732000	42000
180	STTP	229552400	4456400	1275291	6340080	80
190	Reno	265871000	40776000	1399321	12301000	5000
190	Vegas	269193000	44105000	1416805	5441000	82000
190	STTP	228915200	3819680	1204817	11038080	80
200	Reno	265593000	40501000	1327965	12334000	5000
200	Vegas	268748000	43660000	1343740	5244000	107000
200	STTP	229738000	4641920	1148690	7297080	80

REFERENCES

[1] E.W. Knightly and P. Rossaro, "Improving qos through traffic smoothing," in *IFIP IWQoS'96*, 1996.

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

[3] 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.

[4] 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.

[5] Information Sciences Institute J. Postel, "Rfc 793, transmission control protocol," Internet RFC 793, 1981.

[6] 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 IEE Conference on Telecommunications (ICT'98)*, June 1998.

[7] 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.

[8] A. Tanenbaum, *Computer Networks*, Prentice-Hall, 3rd Edition, 1996.

[9] S. Keshav, "A control-theoretic approach to flow control," in *Proceedings of ACM SIGCOMM 1991*, 1991.

[10] R.L. Carter and M.E. Crovella, "Dynamic server selection using bandwidth probing in wide-area networks," Tech. Rep., Computer Science Department, Boston University, 111 Cummington St., Boston MA 02215, USA, March 1996.

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

[12] S. Keshav, *Congestion Control in Computer Networks*, Ph.D. thesis, Department of EECS, UC Berkeley, 1991.

[13] M. Allman and V. Paxson, "On estimating end-to-end network path properties," in *ACM SIGCOMM '99, Cambridge, MA, USA*, September 1999.

[14] R. Wade, M. Kara, and P.M. Dew, "Study of a transport protocol employing bottleneck probing and token bucket flow control," Tech. Rep., September 1999.

[15] 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," Tech. Rep. 99-702, University of Southern California, March 1999.

[16] W. David Kelton Averill M. Law, *Simulation, Modeling and Analysys*, McGraw-Hill International Editions, second edition, 1991.

[17] F. Kelly, A. Maulloo, and D.Tan, "Rate control for communication networks: shadow price proportional fairness and stability.," *Journal of the Operational Research Society*, vol. 49, pp. 237–252, 1998.

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

[19] S. Keshav, "Real: A network simulator," Tech. Rep. 88/472, Department of Computer Science, UC Berkeley, 1998.

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

[21] S. Floyd and J. Mahdavi, "TCP Friendly Web Site," 1999, http://www.psc.edu/networking/tcp_friendly.html.

[22] J. Mahdavi and S. Floyd, "Tcp-friendly unicast rate-based flow control," http://www.psc.edu/networking/papers/tcp_friendly.html, 1997.

[23] B. Ahlgren, M. Björkman, and B. Melander, "Network probing using packet trains," Submitted to Global Internet '99, March 1999.