Available online at www.sciencedirect.com
SCIENCE DIRECT
Computer Communications 29 (2006) 525–537
computer communications
www.elsevier.com/locate/comcom

A simulation-based comparative evaluation of transport protocols for SIP

M. Lulling*, J. Vaughan

Department of Computer Science, University College Cork, Western Road, Cork, Ireland

Available online 10 May 2005

Abstract

Research presented in this paper uses the Network Simulator, ns2, to investigate the direct effects and subsequent consequences associated with the use of different transport protocols in a SIP context. Specifically, we seek to perform a comparative evaluation of UDP, TCP and SCTP based on a model simulating a scenario most likely to benefit from congestion control and error correction mechanisms associated with the reliable transport protocols. The overall aim of this paper is to make a contribution to the information available for the appraisal of such protocols in respect of real world implementations.

Keywords: SIP; Transport; TCP; UDP; SCTP; Performance evaluation

Contents

* Corresponding author. Tel.: +353 21 4902795; fax: +353 21 4274390.
E-mail addresses: m.lulling@cs.ucc.ie (M. Lulling), j.vaughan@cs.ucc.ie (J. Vaughan).

0140-3664/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.comcom.2005.04.006

1. Introduction

This paper presents results from simulation-based experiments to provide information and analysis on the effects of network conditions on Voice over IP (VoIP) signalling traffic and the choice of underlying transport protocol. In this regard the use of SCTP, TCP and UDP in a session initiation protocol (SIP) [1], VoIP signalling context will be compared. As will be detailed in the following sections of this paper, the multiple SIP session scenario present in proxy-to-proxy communication provides an underlying topology that is most likely to benefit from the flow control and congestion control and avoidance features in TCP and SCTP. Results will be presented in terms of delays as well as measured throughput, a performance evaluation gauge first used in this context in preliminary work on the subject [2].

Previous work focussed on the Tahoe, Reno and Sack variants of the TCP transport protocol [3]. It was concluded that Tahoe was completely inappropriate for a SIP signalling context. In this paper, the scope of the research has been extended to include new results for SCTP and UDP as well as the previously omitted Vegas TCP variant. The experimental basis has also been broadened for the simulations using a novel metric to give a further perspective for analysis. Preliminary results comparing TCP Sack to SCTP were presented in [3] and this is comprehensively explored here.

The remainder of this paper is structured as follows. Section 2 discusses transport layer protocols for SIP, specifically TCP, SCTP and UDP. Section 3 introduces the simulations and Section 4 presents the results. Concluding remarks are included in the final section.

2. Transport for SIP

SIP signalling messages are exchanged between IP Telephony users before, during and at the conclusion of a particular IP Telephony session. In this work, the initial messages in session setup are considered. These messages are typically sent from the sender to the receiver via the sender's local SIP proxy and the receiver's local SIP proxy, respectively, as shown in Fig. 1 (provisional responses have been excluded). Subsequent messages are exchanged directly since location information is swapped in the first exchange. Messages exchanged between individual users and their local proxies are characterised as low volume, short-lived, single session messages. Messages from multiple users exchanged between proxies are typically high volume, independent session messages in a long-lived communication session. For this performance evaluation the latter scenario has been focused on, in order to allow the flow control and error correction and recovery mechanisms employed by the TCP and SCTP transport protocols to be used to maximum advantage.

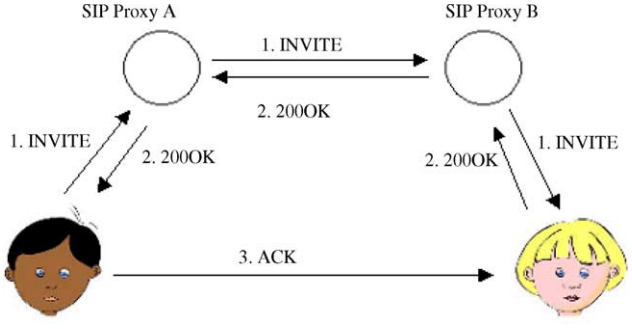

Fig. 1. Typical SIP message sequence for initial call setup.

SIP signalling messages require a delivery guarantee. Due to their nature, SIP messages carry session critical information and thus some mechanism must be in place to ensure that they reach their destination in the event of message corruption or message loss due to network congestion; the latter being the more likely event. UDP does not provide any mechanism in this regard and thus the application layer is responsible for monitoring message status and retransmitting where necessary. Both TCP and SCTP on the other hand take care of this, with the result that the application can 'drop' the message onto the transport layer and 'forget' about it. It would seem, therefore, that a reliable protocol such as TCP or SCTP would be the natural choice; however, both SCTP and TCP have limitations that are particularly pertinent to time sensitive SIP signalling messages. The following subsections will consider SIP transport using the three transport protocols: UDP, TCP and SCTP. For TCP, three variants will be considered, namely the Reno, Vegas and selective Acknowledgement variants.

2.1. SIP over TCP

In the current public Internet, the most widespread transport protocol employed is the transmission control protocol. The reasons for this, historical or otherwise, are outside the scope of this paper, but it is worth noting that a majority portion of competing traffic likely to be encountered on the Internet will be using TCP as a transport protocol. It is also worth noting that TCP is not a concluded standard protocol—there are many different variants with distinguishing differences between most of them, and new versions have been continuously proposed to try to elevate transmission performance.

TCP was not designed with signalling in mind, and has some limitations in this regard. TCP was initially designed to transport large amounts of non-real-time bulk data between two endpoints. A connection is set up between the endpoints and TCP implements flow control and error correction and recovery based on the dynamic behaviour of the end-to-end traffic. Signalling, however, does not typically consist of large amounts of data. SIP messages are usually relatively small (approximately 512 bytes) and SIP uses these messages in a request/response model. Thus, we have small messages that are critically interdependent.

TCP's flow control mechanisms and error correction and recovery mechanisms are not designed for such traffic patterns and TCP's performance is thus not as good as might be expected in this scenario.

Single SIP sessions usually take place between two user agents (UAs), or between UAs and proxies. The limitations of TCP as described are most obvious in this scenario. In particular, connection establishment can present the risk of substantial delays. TCP performs a three-way handshake for connection establishment and requests cannot be sent until this has successfully taken place. In a relatively long-lived connection, the time involved is negligible. However, with a SIP session the duration is usually very short, typically around three or four messages, and connection establishment time becomes significant. Packet loss during the connection establishment handshake can have serious and sometimes intolerable consequences for delay sensitive SIP messages.

Proxy-to-proxy communication, on the other hand, usually involves multiple simultaneous SIP sessions. Bundling these SIP sessions together in the same TCP connection facilitates the use of TCP's flow control and error correction and recovery mechanisms. The disadvantage of connection establishment time is also extenuated in this scenario.

SIP message bundling gives rise to another limitation of TCP. TCP is a byte-oriented protocol, and thus ensures that packets are delivered to the receiving application in order. Because SIP messages are relatively small, the TCP packet is usually made up of a whole SIP message. In this case, the fact that packets are delivered in order is of no advantage to signalling. In the event of packet loss, however, this constraint is a definite disadvantage. If a packet is lost, subsequent successfully delivered packets will not be delivered to the application until the lost packet is retransmitted and received, in order to preserve the order of packets being delivered to the application. This means that packets that contain whole SIP messages, which are independently useful to the application, are unnecessarily delayed to facilitate in-order packet delivery. This is known as head of the line (HOL) blocking.

As noted earlier, TCP is not a concluded protocol. The evolution of different versions of TCP has been provoked by different factors, but improved efficiency and performance in the face of network congestion is common to all. In the very first implementations of TCP, little was done to minimise network congestion. Implementations used cumulative positive acknowledgements and the expiry of a retransmit timer to provide reliability based on a simple go-back-*n* model. Numerous subsequent versions of TCP based on this and, successively, on each other, have been proposed to address this issue. For the purposes of this evaluation, the Reno, Vegas and Sack versions of TCP will be considered and detailed in the following subsections.

2.1.1. TCP Reno

TCP Tahoe, which is also identified as the BSD network release 1.0 (BNR1) version of TCP, added end-system-based traffic control functions to the original TCP standard. New algorithms and refinements were designed to address congestion issues in a network and their adoption was partially influenced by the threat of congestion collapse in the public Internet.

The new algorithms, first mooted by Van Jacobson [4], included:

- Slow Start
- Congestion Avoidance
- Fast Retransmit

Central to these algorithms is the notion of a congestion window, which is used to control the amount of transmitted data. The size of this congestion window relates to the amount of unacknowledged data allowed to be 'in the system' and is changed dynamically during the session in an effort to react to changing network conditions.

Slow Start begins either initially, or in the event of a packet retransmission. The initial congestion window is set to one packet. With each acknowledgement received, the congestion window is incremented by one, thus increasing the window size exponentially. A message sequence diagram illustrates the Slow Start algorithm in Fig. 2.

Once a predefined threshold for the congestion window is reached the sender replaces the Slow Start algorithm with the Congestion Avoidance algorithm. Using this algorithm, the congestion window is increased by a reciprocal of itself for every acknowledgement received, thus growing the window size linearly instead of exponentially.

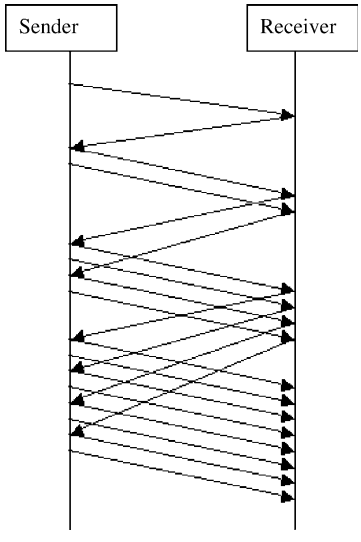

Fig. 2. Slow Start message sequence diagram.

In the event of an out-of-sequence packet being received by the receiver, the packet in question is not acknowledged; instead, a duplicate acknowledgement for the last correctly (i.e. in sequence) received packet is transmitted. After a predefined number of these duplicate acknowledgements are received, usually three, the sender infers that a packet has been lost and retransmits the packet immediately without waiting for the expiry of a retransmission timer. This is known as the Fast Retransmit algorithm.

The TCP Reno implementation, also identified as the BSD network release 2.0 (BNR2) version of TCP, in addition to the enhancements incorporated into Tahoe, includes the fast recovery algorithm. This is an extension of the Fast Retransmit algorithm, which optimises Reno for the case when a single packet is dropped from a window of data.

Instead of performing Slow Start after the retransmission of a lost packet, TCP Reno uses the fast recovery algorithm. Essentially, the congestion window is reduced to half its value plus the duplicate acknowledgement threshold (which, as noted earlier, is usually three). The congestion window is temporarily incremented with every subsequent duplicate acknowledgment received until the acknowledgement of the retransmitted packet is received. Once this 'recovery' acknowledgement has been received, the congestion window is reset to half of its initial value on entering the fast recovery state and the sender resumes with the Congestion Avoidance algorithm.

Reno also supports the use of delayed Acknowledgement, which, instead of acknowledging every packet, acknowledges every other packet. In many implementations, however, this feature is turned off. For the purposes of this simulation study this delayed Acknowledgement option will not be used.

The algorithms for TCP Reno, which include those for Tahoe, outlined above are described in full detail in [5].

2.1.2. TCP Vegas

Due to the nature of the congestion control mechanism adopted by TCP Reno, the congestion window size continues increasing until packet loss is encountered. The subsequent reduction in size of the congestion window can cause degradation in the connection throughput. This less than desirable consequence transpires because network congestion can only be detected by Reno with packet loss. If the congestion window size is appropriately controlled such that packet loss does not occur, the throughput degradation can be avoided. This can only be achieved with some alternative means of detecting network congestion.

TCP Vegas [6,7] takes such a different approach and uses the measured round trip time (RTT) to accurately calculate the amount of data packets that the sender can send to avoid packet losses. If the observed RTT becomes larger, TCP Vegas infers that the network is becoming congested and reduces its congestion window size. Conversely, if observed RTTs become smaller, Vegas allows the congestion window size to increase. Ideally, the congestion window converges to an appropriate value.

TCP Vegas also has other less radical differences: in addition to Vegas' modification to Reno's Congestion Avoidance algorithm as above, the Slow Start mechanism is also altered. Instead of incrementing the congestion window with every ACK packet received during the Slow Start phase, Vegas increments the congestion window with every other ACK packet.

With these modifications TCP Vegas is expected to perform substantially better than TCP Reno. According to [6], TCP Vegas can achieve upwards of 40% higher throughput than TCP Reno, and this is supported with related simulation and implementation experiments. However, recent research has shown that TCP Vegas suffers a serious disadvantage when in competition with TCP Reno, unless parameter adjustment is made [8]. This leads potential implementers to the inescapable conclusion that TCP Vegas cannot compete effectively on the public Internet and would produce poor performance results in that environment.

Vegas is not widely implemented and is generally regarded as being an experimental transport protocol.

2.1.3. TCP Sack

While Reno is the most widely used TCP variant, it has problems other than those intended to be addressed by TCP Vegas. The method of estimating available bandwidth by detecting packet loss employed by Reno is prone to causing network congestion and gives rise to unstable bandwidth usage. This is compounded by the fact that Reno is optimised for single packet loss from a window of data; depending on the network topology, single packet loss or very small numbers of packets lost from a window of data can be dealt with by the Fast Retransmit algorithm. Multiple packets lost from a window of data, however, can have significant effects on throughput, mainly due to the fact that the conditions do not exist to facilitate Fast Retransmit and the expiry of a timer is required for retransmission of missing data.

The selective Acknowledgement strategy was proposed to address the problems presented by multiple packet loss from a window of data. Initially defined in [9], it was updated and modified in [10]. Essentially, the Sack version of TCP provides a mechanism whereby the data receiver can inform the sender of all packets successfully received rather than simply acknowledging the last in-order packet correctly received. This prevents the retransmission of packets that have been successfully received but not acknowledged. The Sack option is negotiated during the connection establishment handshake and if both endpoints support the option, then both ends will use it in the ensuing session.

When an out of order packet arrives at the receiver, a duplicate ACK for the last in-order packet received is returned. When using Sack, a pair of sequence numbers

describing the out of order data block received is contained in the ACK. This allows the sender to build a table of correctly received packets, thus substantially reducing the retransmission of redundant data blocks. Duplicate ACKs, however, are still treated in a similar way to Reno: the data recovery mechanisms commence after a predetermined number of duplicate acknowledgements are received, typically three.

2.2. SIP over UDP

The user datagram protocol, UDP, is a connectionless transport protocol that does not provide any guarantee of message delivery [11]. Thus, the limitations associated with TCP discussed above are not relevant to UDP. The fact that the protocol is connectionless means that connection establishment time is not a factor, an issue that is particularly relevant to the single SIP session scenario typically seen in UA to UA, or UA to proxy communication.

The fact that message delivery is not guaranteed means that the application layer must provide a mechanism for dealing with this. The SIP specification details a retransmission policy that is used by SIP applications using UDP as a transport layer protocol. A retransmission timer is associated with each SIP message and the message is retransmitted on expiry of the timer. The underlying assumption for the retransmission is that both the number of retransmissions and the SIP messages themselves are relatively small, and are not considered a potential source of network congestion. Thus, no element of congestion control is included in the retransmission mechanism, and as a result is much more aggressive in this respect than TCP.

The timer used for SIP retransmissions starts at 500 ms by default and doubles with every retransmission. This gives retransmission intervals of 1, 2, 4, 8, 16 and 32 s; if no response is received after the sixth retransmission, the destination is considered unreachable.

The limitations associated with TCP in the multiple SIP session scenario, typically present in proxy-to-proxy communication, are not present when UDP is used. Other complications arise; however, the most serious of which is that the same retransmission times are used for all messages and in such a high-traffic environment this can be a source of network congestion caused by a retransmission storm. UDP does not provide congestion information and the aggressive retransmission policy employed by SIP applications when using UDP may not always be advantageous. Supporting retransmissions at the application layer is also very expensive in terms of overhead, and a lot of state must be maintained. This raises scalability concerns which are very pertinent in today's evolving Internet environment.

2.3. SIP over SCTP

The stream control transmission protocol, SCTP, is a reliable end-to-end transport layer protocol [12], and while support for TCP and UDP is included in the core SIP specification, support for SCTP is currently work in progress at the IETF [13]. Like TCP, SCTP implements flow control and error correction and recovery mechanisms to dynamically adapt to network conditions and provide a guarantee of message delivery. In contrast to TCP, SCTP was designed with signalling in mind and was intended for use as a reliable transport service between two endpoints exchanging signalling messages. It has a number of useful features in this regard. Of course, it is envisaged that SCTP will provide benefits to other applications outside this specific domain, but such a discussion is outside the scope of this paper.

The most important difference between SCTP and TCP from our point of view is the fact that it is a message-oriented protocol rather than a byte stream one. SCTP supports the framing of individual messages by defining data chunks within the payload. In a SIP signalling context, each data chunk would contain a whole SIP message. SCTP has a notion of 'streams' to which the data chunks belong, the idea being that individual streams identify related messages. The SIP protocol, however, already contains the information necessary for demultiplexing SIP traffic at endpoints and so all messages should normally be set to a default stream.

SCTP also has the option to provide an unordered delivery service as well as an ordered one. Using the unordered delivery service, which allows out-of-sequence data chunks to be passed to the application as soon as they arrive, the most serious of TCP's disadvantages is avoided: head of the line blocking. As noted earlier, SIP messages are independently useful to the application and the delays associated with an ordered delivery service are both unnecessary and undesirable.

The algorithms used for flow and congestion control and error correction and recovery are the same as those used for TCP Reno, as outlined in Section 2.1.1 and detailed in [5]. Also, in a similar way to TCP Sack, SCTP uses selective acknowledgments to reduce the retransmission of redundant data. However, Sacks are used to acknowledge and provide information on individual data chunks received rather than whole packets.

Disadvantages in common with TCP do exist, nevertheless. Initial connection establishment is achieved by means of a four-way handshake and this can present the same drawbacks as with TCP. However, data is allowed to be sent with the third handshake message, presenting no additional connection establishment delay. Network congestion is also detected by packet loss alone; the negative aspects of which have already been discussed.

SCTP has other features and enhancements as a transport protocol such as multi-streaming, multi-homing and improved security, but these are not immediately relevant to this document.

3. Simulations

Recent work [3] compared the Tahoe, Reno and Sack variants of TCP using simulation experiments carried out on the network simulator [14], and results for Reno and Sack from this study are included here to form part of this comparative evaluation. For the research presented in this paper, the scope of the simulation experiments has been extended to cover the UDP and SCTP transport protocols and the further metric of system throughput. Other related work in this area has been conducted by Camarillo et al. [15], though the primary focus in their work was an evaluation of the stream control transmission protocol.

The network topology is illustrated in Fig. 3; nodes 1 and 2 are buffer-limited droptail routers and all other nodes are endpoints. These simulations examine the behaviour of SIP traffic between two proxies (nodes 0 and 3) using the different transport protocols and variants. Nodes 4 and 5 are used to provide competing cross-traffic as appropriate. The link between nodes 1 and 2 is the only bottleneck link, and bandwidth values have been chosen accordingly. While the number of nodes used in the simulations is relatively small—in practice one could expect up to 50 hops between such proxies—the simplified model with a single bottleneck link allows meaningful observations and conclusions can be drawn without compromising the underlying characteristics of that which is found in practice. The delay values were chosen to correspond to an overall delay of 45 ms between the proxies, which reflects a realistic transmission delay likely to be encountered between a hypothetical SIP server in Europe and a corresponding one in the United States. Similar configurations have underpinned related work [2,3,15].

For traffic generation, the simulations use a stationary Poisson model to generate the arrival times of 512-byte session establishment requests at node 0. This models a typical SIP session establishment INVITE request arriving from a user to a SIP proxy. Such requests are assumed to be independent of one another. These requests are immediately forwarded to another SIP proxy at node 3. Node 3 would typically respond with a provisional 100 Trying response on successful receipt of the request. Provisional responses are not considered in the results. The Poisson model is commonly used to model telephone call arrivals [16], and all the underlying requirements for this model are satisfied in these simulations. A non-stationary model is usually applied with the assumption that call arrival rates will vary depending on the time of day or indeed the day of the week.

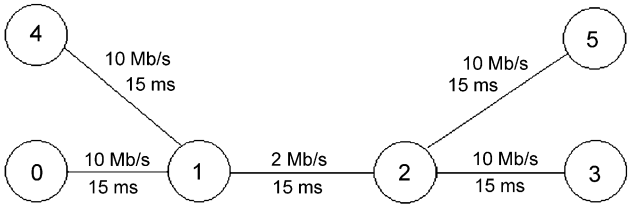

Fig. 3. Network topology.

However, the relatively short simulation time negates the dependence on these covariates, and allows the application of the stationary Poisson model for traffic generation. Other commonly used models, such as non-Poissonian models typically associated with Internet traffic generation in simulations, are not appropriate to the restricted context of SIP signalling.

This configuration models a hypothetical outbound proxy for one service provider and an inbound proxy for another service provider. Users would typically communicate with such a proxy in the early stages of call establishment.

To perform the comparison of transport layer protocols for SIP signalling, a number of experiments have been designed to highlight the strengths and weaknesses of the different transport protocols and variants. Some of these experiments were used similarly in [3]. In all experiments, the SIP traffic is generated in an identical manner. Individual SIP requests are assumed to be independent of one another and are generated at node 0 at exponentially distributed intervals. Requests are generated at a rate of 160/s, which corresponds to a link utilisation of approximately 33% on the bottleneck link. It was found that with higher rates, the TCP throughput in particular was unable to contend with the traffic being generated, and any unfavourable behaviour in the system resulted in destabilisation and dramatically increasing delays.

3.1. Induced packet loss

In order to measure and evaluate the delays and delaying effects of packet loss on the system, packets are explicitly dropped from node 1. While this does not simulate a realistic packet loss scenario that might be observed in practice, it provides valuable statistics on the behaviour in this controlled environment. The simulation is run 10 times for each of the five transport protocols or variants, dropping increasing numbers of consecutive packets with each different run. The time at which each message is generated by the application at node 0 and the time at which this message is passed to the application at node 3 is recorded and the results which are discussed in Section 4 are based on the delays derived from the difference between these times.

3.2. Random packet loss

This scenario is one that more accurately reflects a realistic network environment. Five simulations are run in this case for each of the five transport protocols or variants. Random packet loss percentages of between 0.1 and 0.5% (in 0.1% intervals) are simulated at node 1 with uniform distribution. These choices of loss percentages correspond conservatively to network conditions likely to be encountered by an IP Telephony call [17]. For example, a leading Internet service provider in the United States cites 0.7% as a maximum upper bound on its long-term loss probability [18]. As with the previous experiment, the time at which

each message is generated by the application at node 0 and the time at which this message is passed to the application at node 3 is recorded, and the difference between these represents the delay encountered.

3.3. Competing traffic

This final delay experiment is designed to simulate the effects of cross-traffic generated between nodes 4 and 5, and thus providing competition for bandwidth on the bottleneck link between nodes 1 and 2. This gives rise to queuing delays as well as potential packet loss. The scenario of node 4 transferring a large file by FTP to node 5 is simulated. The transfer time within the simulation is much less than the overall time considered for the simulation. TCP Reno is used exclusively as the transport protocol for the competing traffic in all simulations, since this is the most common transport protocol employed for this purpose. Delays encountered by the SIP messages are again measured as described in the two previous experiments.

3.4. Throughput analysis

In a departure from traditional performance measures appropriate to these experiments, the overall system throughput is considered in the face of continuous competition on the bottleneck as a metric for the comparison. A heterogeneous competition environment is simulated, involving the SIP traffic competing for bandwidth on the bottleneck link with a simulated FTP application sending bulk data between nodes 4 and 5. FTP uses TCP as a transport protocol and the simulation has been run with the Reno, Vegas and Sack variants. SIP traffic competing against background traffic from an FTP application using TCP Reno is arguably the most interesting scenario due to the fact that this resembles that which is most likely to be encountered on the public Internet at present.

Adding a variable of buffer size at node 1, i.e. varying the available buffer for the bottleneck link, provides another dimension to this performance evaluation. The simulations have been run with buffer sizes of 5, 20, 50, 100, 150, 200 and 250 packets at node 1. The default setting for the simulator is a buffer length of 50 packets, which is generally accepted to be a good approximation of average buffer size; in practice, however, buffer sizes vary substantially.

4. Results

In this section, the results from the experiments detailed in Section 3 are presented. The results from the different experiments will be individually discussed, and the overall comparison will be dealt with in Section 5. Essentially, the first three experiments consider delays and delaying effects experienced by the SIP messages, the final experiment deal with throughput and is a metric that is not usually applied to this context.

Note that the Internet is not governed by any existing standards or recommendations regarding transmission delays and none are likely to emerge in the near future [19]. Whereas Eyers and Schulzrinne [19] also investigate IP Telephony signalling delays, they do so only in relation to the use of UDP as a transport protocol for SIP. Their scenario involves call setup within the United States and does not extend to intercontinental calls. This means that, although the delay measurements are important and interesting in their own right, it is difficult to interpret them in the context of acceptable Internet call setup delays. This can be somewhat compensated for by relating the figures to the public switched telephone network (PSTN), for which strong recommendations are available, in particular with respect to quality of service and call setup delay. The International telecommunications union recommendation E.721 [20] recommends an average setup delay of 3 s for local calls and 8 s for international calls using the PSTN. These limits are frequently exceeded on the PSTN. Furthermore, just one part of the total signalling path is simulated, so that the overall setup delay may be much greater. Therefore, it is inferred that it is more appropriate to use the E.721 recommended mean values as maximum upper bounds for delays in this scenario.

A tighter constraint applies when Internet to PSTN connection is considered. In this situation, the ISDN switches used for such an interconnection may abandon a call if a reply from a setup attempt is not received within 2 s [21]. These simulation results show that this constraint may be very difficult to satisfy under certain conditions.

4.1. Induced packet loss

Fig. 4 graphs the results from the experiment, with five consecutive packets being explicitly dropped in this case. The axis of abscissae displays the message sequence numbers corresponding to individual SIP messages and

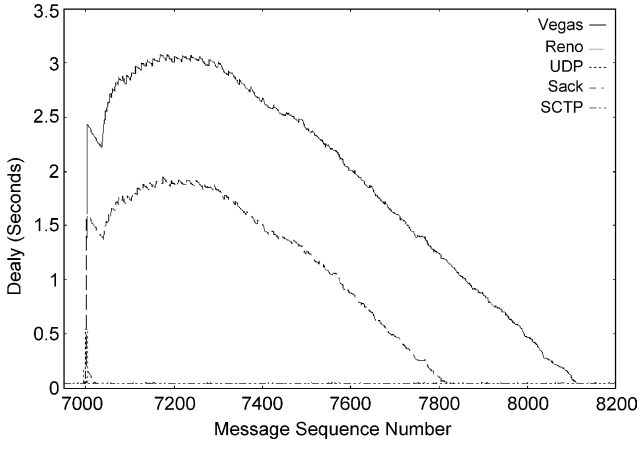

Fig. 4. Five consecutively dropped packets.

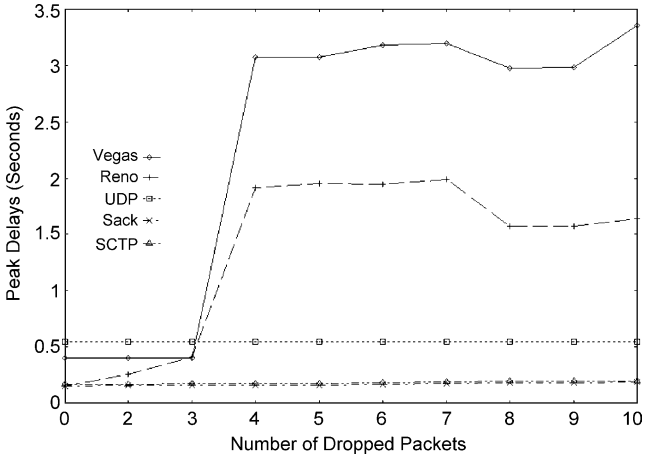

Fig. 5. Peak delays.

the ordinate axis displays the delays encountered in seconds. The delays and delaying effects encountered by SIP messages being bundled into a TCP Vegas connection are dramatic. The results from TCP Reno give a similar impression, albeit to a lesser extent. Due to the extreme nature of the delays from TCP Reno and TCP Vegas, the results from SCTP, UDP and TCP Sack are not very obvious and are nearly rendered inconsequential in comparison. The following graphs, Figs. 5 and 6, give a more useful view of the consequential delays and delaying effects.

Fig. 5 graphs the peak delays encountered by each of the different transport protocols or variants at each of the 10 different levels of consecutive packet loss. The axis of abscissae displays the numbers of consecutive packets explicitly dropped. The axis of ordinates displays the peak delays encountered in seconds. These results provide a very useful overview of what magnitude of delays can be encountered. All protocols seem to behave reasonably up to the loss of three consecutive packets. At higher packet losses there is a distinct performance demarcation: SCTP and TCP Sack both perform very well, with very small peak delays. SIP/UDP does not perform quite as well, due to its total reliance on retransmission timers, but is still within

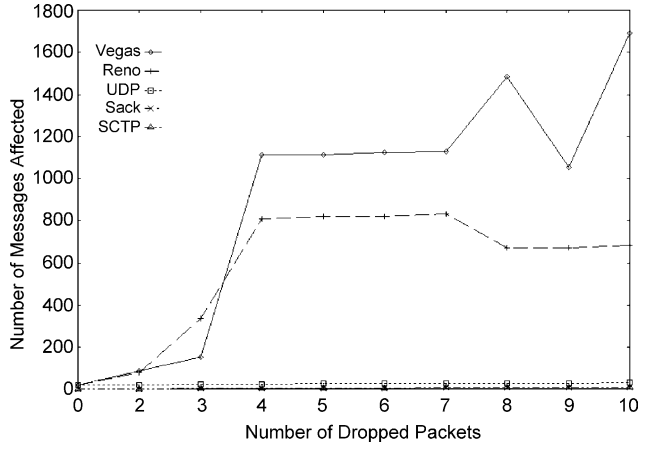

Fig. 6. Messages affected.

acceptable levels. TCP Reno, and particularly TCP Vegas, both raise questions about their performance in this instance.

Fig. 6 graphs the overall numbers of messages affected by the initial loss and retransmission of the consecutively dropped packets. One of the underlying assumptions of the system is that the SIP messages have arrived independently at the SIP proxy at node 0. The fact that they are bundled into a single long-lived connection for SCTP and the TCP variants, compromises their independence to a degree and it can be seen from the graph that subsequent unrelated messages are delayed as a direct consequence of the loss of the packet(s) in question. As can be seen, UDP, SCTP and TCP Sack perform, admirably, with only a very minimum number of unrelated messages encountering delays. Again TCP Reno and, particularly, TCP Vegas give cause for concern. Large numbers, in the order of hundreds, of unrelated messages encounter delays.

Overall SCTP, UDP and TCP Sack perform adequately under the conditions. TCP Reno and TCP Vegas, however, exhibit worrying behaviour giving rise to large peak delays and large numbers of subsequent unrelated messages encountering consequential delaying effects. This would be considered unacceptable for SIP signalling.

4.2. Random packet loss

Fig. 7 displays the results for the random packet loss experiment with a loss rate of 0.3%. For clarity the results have been individually graphed for each transport protocol or variant (a)–(e). Due to space constraints, results for the other experiments with loss rates of 0.1, 0.2, 0.4 and 0.5% have not been included here; however, the discussion has been generalised to include them.

The 0.3% packet loss rate was chosen to be graphed since it is only at this rate, or lower, that the most stringent delay bound of 2 s is met by all protocols. At higher loss rates, this bound is exceeded by varying degrees with protocols TCP Sack, TCP Reno and SCTP. UDP and TCP Vegas consistently outperform the other protocols considered. In the case of UDP, this was expected: message independence is never compromised in the connection and packet loss is dealt with exclusively by retransmission. UDP rarely needs a second retransmission to deliver the message successfully.

The performance of TCP Vegas in this experiment is somewhat contradictory when compared with the other experiments. However, when TCP Vegas was first proposed, the authors claimed a performance increase of 31–71% over TCP Reno in terms of throughput in a homogeneous environment. While these results cannot be directly related to the particular context of SIP signalling, it would be reasonable to anticipate some performance improvement in this regard. What has been observed is a marked performance improvement in this experiment, mainly due to the fact that TCP Vegas does not encounter any competing traffic here; in the other experiments,

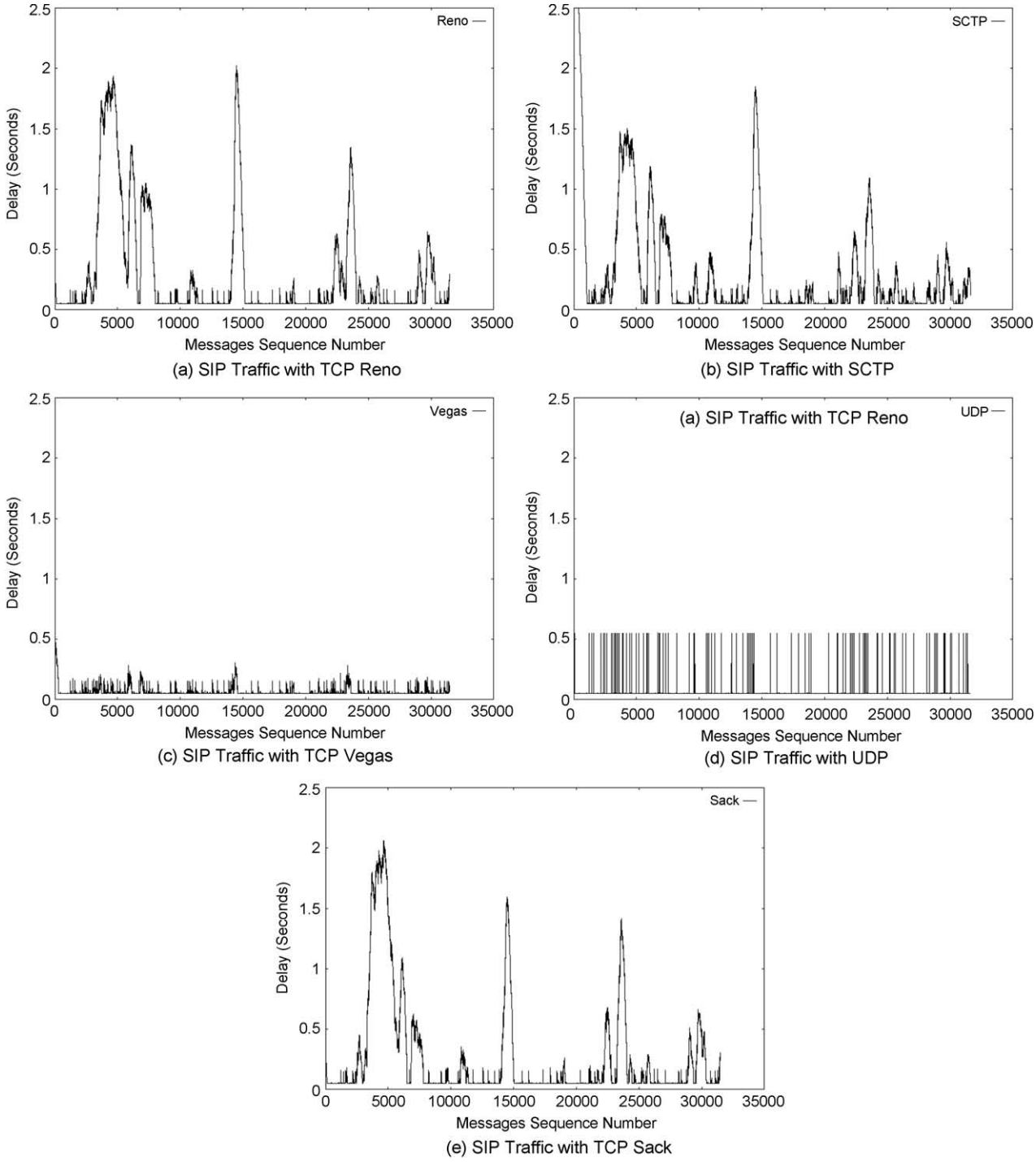

Fig. 7. 0.3% Random packet loss delays.

particularly those with competing traffic, TCP Vegas suffers from an unfairness problem as noted earlier in Section 2.

Fig. 8 presents a summary of the results from this experiment: the axis of abscissae displays the packet loss rate while the axis of ordinates displays the mean overall delay encountered by the SIP messages. The worst result, here, presented by TCP Reno at the 0.5% loss rate is in fact the only one which breaches the stringent 2s delay bound. While this value is normally an absolute one and not usually applied to averages, it is useful in this context. TCP Sack and SCTP, while offering a better performance and critically within the 2s bound, still display an undesirable increasing trend of overall mean delays. Both UDP and TCP Vegas on the other hand outperform the other protocols/variants with a substantially increasing margin as the packet loss rates increase. Indeed, the increasing loss rates do not appear to affect UDP at all, and only affect Vegas to a very minimal degree.

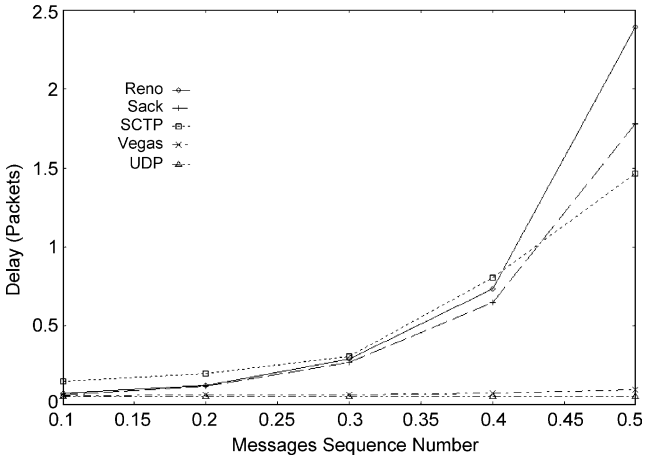

Fig. 8. Mean delay per packet loss percentage.

4.3. Competing traffic

In this experiment, competition on the bottleneck link is simulated for a fixed period during the overall simulation. This period, where FTP traffic between nodes 4 and 5 is competing for bandwidth on the bottleneck link, is clearly obvious in all at the graphs. The delays encountered by the SIP messages are again graphed against the individual message sequence number (Fig. 9).

The four graphs shown deal with SCTP, UDP and the Sack and Reno variants of TCP. TCP Vegas results have not been graphed; using TCP Vegas, the system destabilised with dramatically increasing delays for the entire duration of the competition traffic. It was immediately clear that TCP Vegas was not a feasible transport protocol in this situation.

Of the results graphed, TCP Reno stands out. Peak delays, while substantially bigger than those in the other graphs, are still relatively short-lived and the system recovers each time. These peak delays do breach the 2s limit, however, giving some cause for concern. In all other results graphed, the peak delays are all comfortably within any potentially appropriate limits that might be chosen to apply. SCTP and UDP both perform very well with only a minimum of delays and a clearly stable system. TCP Sack's results also display stable characteristics and while peak delays are encountered more frequently, subsequent delaying effects appear to be minimal.

The obvious result from this experiment is that TCP Sack is the only variant of TCP that has a performance

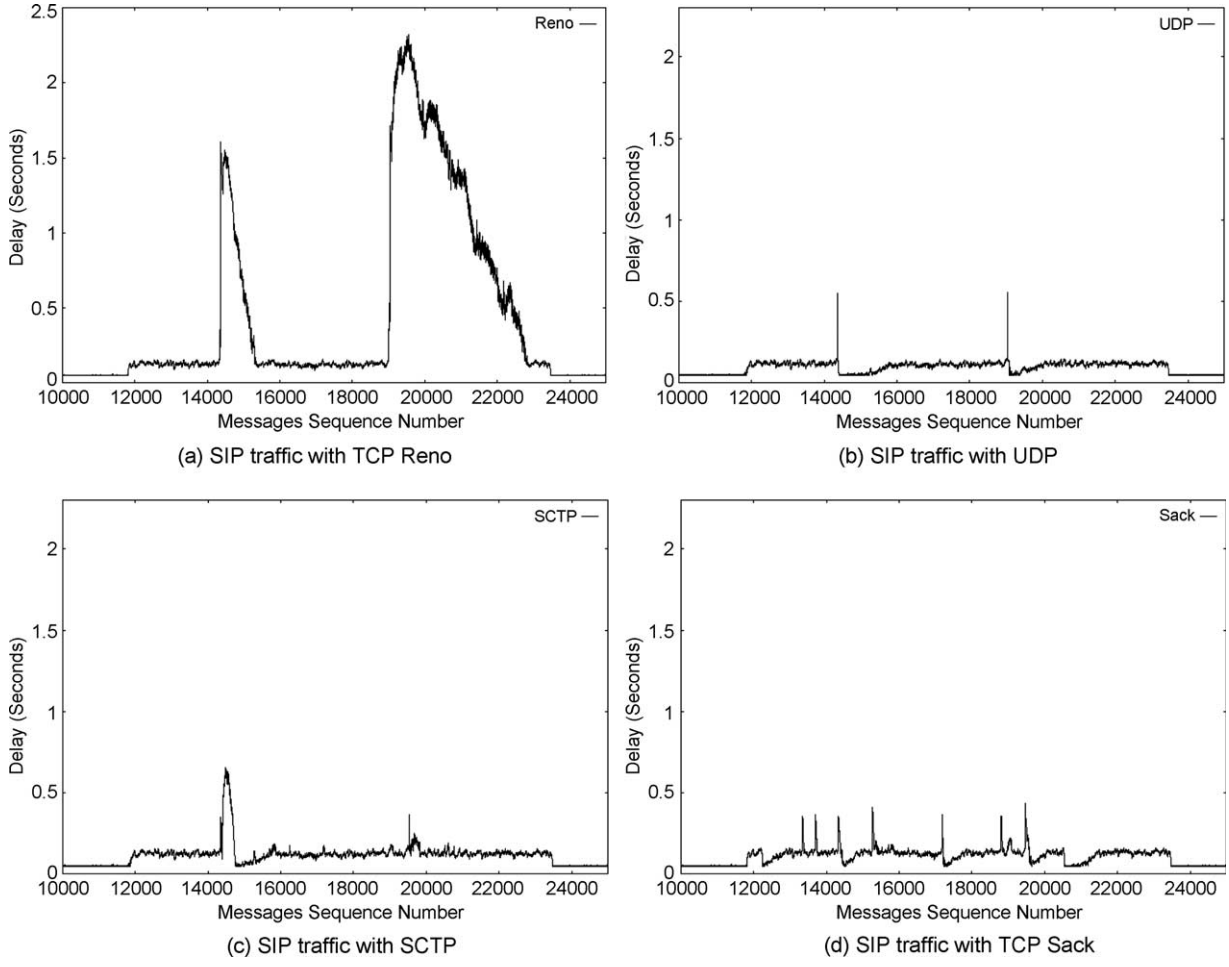

Fig. 9. Delays encountered by SIP traffic when in competition with FTP TCP Reno.

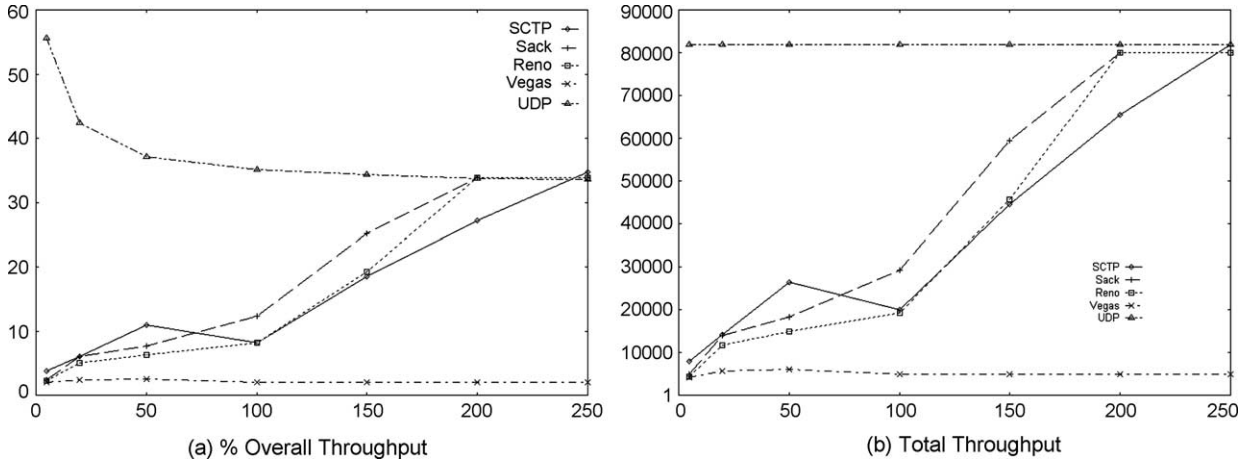

Fig. 10. Throughputs achieved by the SIP application with different transport protocols.

commensurate with that which is observed when using SCTP and UDP.

4.4. Throughputs

For this experiment, a heterogeneous competition environment has been simulated, and the percentage of overall throughput achieved has been graphed, as well as the total throughput achieved by the SIP application, in Fig. 10(a) and (b). In these two figures, the competition at the bottleneck link is provided by a simulated FTP traffic using TCP Reno, which, as previously stated, is the most apposite scenario for modelling the Internet. It would be reasonable to expect that the percentage overall throughput achieved in a well-balanced system should remain relatively constant as the buffer size is varied. What is observed, however, is that while SIP with TCP Vegas behaves in this way, SIP with TCP Reno, TCP Sack and SCTP do not, and their performance in this respect has a marked deterioration as the buffer size is reduced. SIP with UDP, conversely, displays a performance improvement at lower buffer sizes. From Fig. 10(b), it can be seen that this is because the total throughput achieved by the UDP application does not vary, even at the lowest buffer size: while the overall throughput may be much reduced at small buffer sizes, the throughput is unaffected when using UDP. This resilience is partially due to the aggressive retransmission policy employed by SIP applications using UDP, as well as the lack of congestion control functionality. Another pertinent factor is that each message in the UDP application is totally independent: loss of a particular message has no effect on any other message, contrary to the situation with TCP and SCTP.

In both figures, TCP Sack displays a marginal performance advantage over both TCP Reno and SCTP, most marked at the intermediate buffer sizes. This advantage, however, is much less than expected, particularly at the lower buffer sizes where the increased level of packet loss should favour the selective acknowledgement option employed by both TCP Sack and SCTP.

Immediately obvious from the results is the significant disadvantage of using the TCP Vegas variant. The throughput suffers dramatically as a result of competition. The fact that the percentage of overall throughput achieved is pretty consistent across the various buffer sizes is small consolation and shows that while the system may be balanced in this scenario, it is not necessarily fair. Even when Vegas competes with itself, the results have shown that the overall throughput values achieved in the system are lower indicating that maximising resource usage presents a problem for Vegas. Dynamic parameter adjustment, as suggested in [8], was not considered, since it is unlikely that such an implementation will be encountered on the public Internet at present.

Other results, presenting the SIP traffic with competition from background traffic of FTP using TCP Vegas and Sack, have not been graphed and are presented in Tables 1 and 2. These types of background traffic are not likely to be encountered in the public Internet and have only been included here for completeness. The clear performance advantage displayed by the use of UDP is also reflected in these results, as is the disadvantage of using TCP Vegas.

Table 1
Mean throughput for SIP vs. FTP TCP Sack

Buffer size	SIP/TCP Vegas	SIP/TCP Reno	SIP/TCP Sack	SIP/ UDP	SIP/ SCTP
5	1845	5082	4826	82,015	6559
20	6359	6353	12,925	82,008	16,474
50	5480	11,710	18,186	82,013	21,994
100	4871	16,999	22,632	82,011	23,915
150	4871	44,044	53,375	82,015	44,611
200	4871	80,098	80,098	82,014	65,495
150	4871	80,098	80,098	82,015	82,015

Table 2
Mean throughput for SIP vs. FTP TCP Vegas

Buffer size	SIP/TCP Vegas	SIP/TCP Reno	SIP/TCP Sack	SIP/ UDP	SIP/ SCTP
5	21,321	4161	14,195	82,015	13,808
20	50,849	81,834	81,834	82,015	82,015
50	50,849	81,833	81,833	82,015	82,015
100	50,849	81,833	81,833	82,015	82,015
150	50,849	81,833	81,833	82,015	82,015
200	50,849	81,833	81,833	82,015	82,015
150	50,849	81,833	81,833	82,015	82,015

5. Conclusion

The results from experiments presented here are intended to be used as a basis for an objective comparison of different transport protocols in the context of SIP signalling. Specifically, the stream control transmission protocol (SCTP), the user datagram protocol (UDP) and the Reno, Vegas and Sack variants of the transmission control protocol were considered. These experiments were conducted and the results were derived from a simulated environment. A modelling and simulation approach to performance evaluation was deemed to be the most viable alternative to a full-scale implementation with all the obvious advantages of flexibility and rapid adjustment.

The underlying network topology models a proxy-to-proxy SIP signalling context. This was specifically chosen so as to offer maximum advantage to the long-lived connections employed by TCP and SCTP. Bundling multiple SIP messages into a single end-to-end connection facilitates the use of both TCP and SCTP's end-system-based traffic control functionality.

Four different simulation experiments were conducted, giving results in terms of both throughputs and delays on which to base this evaluation. The first three experiments, concerned with using delays as a performance metric, not only examine the direct effects of differing network conditions on particular SIP messages, but also demonstrate the effect on subsequent unrelated SIP messages. By virtue of being bundled into a single connection, and having flow control and congestion control and avoidance mechanisms imposed on the aggregate of messages within this connection, SIP messages, which are assumed to have arrived at a proxy independently of one another, have had this independence compromised to a degree. TCP in particular, being a byte-stream-oriented protocol with the known problem of HOL, not only induces delays on unrelated SIP messages, but also affects the overall throughput of the connection.

The first experiment, with induced consecutive packet loss, demonstrated clear advantages for TCP Sack and SCTP. UDP also performed well, particularly at the higher numbers of lost packets. TCP Vegas and, to an only slightly lesser extent, TCP Reno display disappointing performance results. Not only are peak delays quite large for individual SIP messages, but subsequent induced delays are imposed on a very large number of unrelated packets.

The random packet loss experiment gave interesting results: very little separated TCP Sack, TCP Reno and SCTP results in terms of performance. As the loss rate increased, TCP clearly had the worst results of the protocols considered. Crucially at the 0.5% loss rate, the mean delay encountered during the simulation exceeded 2 s, a stringent bound described in Section 4. All three protocols exhibited an undesirable but unsurprising trend of increasing mean delays with increasing loss rates. UDP and TCP Vegas, on the other hand, both had excellent results with low peak delays, low mean delays and relatively inconsequential subsequent induced message delays.

The third experiment dealing with a competing traffic environment, and the final one dealing with message delays as a metric, gave the clear result that TCP Sack was the only variant of TCP that could compete with SCTP and UDP. TCP Reno results were poor in comparison and TCP Vegas failed to cope with the competition environment completely.

The fourth experiment in a departure from previous ones, dealt with throughput as a metric. Of the protocols considered, none could compete with the clear performance advantage presented by the use of UDP. The aggressive retransmission policy employed by SIP applications using UDP could contribute to network congestion; however, no such effect was observed in these simulations. The resilience of the application using UDP in the face of the substantial packet loss and a hostile competition environment was remarkable.

As expected, the results have shown that TCP Sack offers the best performance of the different TCP variants considered. The use of a selective acknowledgement option is a definite advantage. TCP Vegas displayed serious problems in certain scenarios and cannot be recommended as a transport protocol for SIP. TCP Reno, while performing better than TCP Vegas, also gave some cause for concern; however, as also detailed in [6], while the advantage offered by TCP Sack over TCP Reno was clear, its magnitude was insufficient to be conclusive. One of the inherent features of TCP Sack is that it can be used in conjunction with TCP Reno, however; the support of Sack is queried and established in the initial connection establishment handshake, and in the event of TCP Sack not being supported, the session can still be conducted using TCP Reno. This means that interoperability issues will not be relevant and there will be no disadvantage in using TCP Sack in that regard.

The performance of SCTP was not as good as might have been expected. Indeed, any perceived difference between SCTP and TCP Sack was so small as to be almost inconsequential. It was expected that SCTP's unordered data chunk delivery option would offer a distinct advantage over TCP in terms of delays, by eliminating HOL blocking. This was not the case and the results lead to the conclusion that the limiting factor in terms of delays for TCP is not the fact that it is a byte-stream-oriented protocol with HOL blocking problems,

but rather the imposition of its end-system-based flow control and congestion control and avoidance mechanisms on an aggregate of messages within a session.

UDP's consistently good performance across all experiments is very obvious. This does come at a cost, however, complexity and overhead at the application layer is undesirable and both are a consequence of using UDP for SIP applications. This notwithstanding, UDP is the most common transport layer protocol employed by SIP applications today and on the basis of the results presented here, the use of either TCP or SCTP as a viable alternative cannot be authoritatively advocated.

In conclusion, it was found that TCP Sack and SCTP are the best options for a reliable transport protocol for SIP traffic. Differences in performance were insufficient to conclude that one was better than the other. Neither protocol offered sufficient performance advantage to allow the conclusion that it should be used in preference to UDP, currently the most popular transport protocol for SIP.

Acknowledgements

This research was funded by the Boole Centre for Research in Informatics (BCRI) at University College Cork.

References

[1] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, SIP: Session Initiation Protocol, IETF RFC 3261, 2002.
[2] M. Lulling, J. Vaughan, An investigation of TCP throughput variation for SIP traffic, in: Proceedings of the International Symposium on Information and Communication Technologies, Dublin, Ireland, 2003, pp. 584–589.
[3] M. Lulling, J. Vaughan, A simulation based performance evaluation of Tahoe, Reno and sack TCP as appropriate transport protocols for SIP, Computer Communications Journal 27 (2004) 1585–1593.
[4] V. Jacobson, Congestion avoidance and control, in: Proceedings of ACM SIGCOMM '88, 1988, pp. 319–329.
[5] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, IETF RFC 2581, 1999.
[6] L.S. Brakmo, S.W. O'Malley, L.L. Peterson, TCP Vegas: new techniques for congestion detection and avoidance, in: Proceedings of ACM SIGCOMM '94, 1994, pp. 24–35.
[7] L.S. Brakmo, L.L. Peterson, TCP Vegas end to end congestion avoidance on a global Internet, IEEE Journal on Selected Areas in Communications 13 (1995) 1465–1480.
[8] Y.C. Lai, C.L. Yao, Performance comparison between TCP Reno and TCP Vegas, Computer Communications Journal 25 (2002) 1765–1773.
[9] V. Jacobson, R.T. Braden, TCP Extensions for Long-Delay Paths, IETF RFC 1072, 1988.
[10] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgement Options, IETF RFC 2018, 1996.
[11] J. Postel, User Datagram Protocol, IETF STD 0006, 1980.
[12] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, Stream Control Transmission Protocol, IETF RFC 2960, 2000.
[13] J. Rosenberg, H. Schulzrinne, G. Camarillo. The Stream Control Transmission Protocol as a Transport for the Session Initiation Protocol, IETF Internet Draft (Work in Progress).
[14] The Network Simulator—ns2 (http://www.isi.edu/nsnam/ns).
[15] G. Camarillo, H. Schulzrinne, R. Kantola, Evaluation of transport protocols for the session initiation protocol, IEEE Network 17 (5) (2003) 40–46.
[16] L.D. Brown, L.H. Zhao, A test for the Poisson distribution, Sankhya: The Indian Journal of Statistics 64 (2002) 611–625.
[17] W. Jiang, H. Schulzrinne, Assessment of VoIP service availability in the current Internet, Proceedings of the Passive and Active Measurement Workshop (PAM), La Jolla, CA, April 2003.
[18] http://ipnetwork.bgtmo.ip.att.net/pws/averages.html (viewed 23/09/2004).
[19] T. Eyers, H. Schulzrinne, Predicting Internet telephony call setup delay, Proceedings of the 1st IP-Telephony Workshop (IPtel 2000), Berlin, Germany, 2000.
[20] International Telecommunications Union, Network grade of service parameters and target values for circuit switched services in evolving isdn, ITU-T Recommendation E.721, 1999.
[21] Bellcore Lssgr: switching system generic requirements for call control using the integrated services digital network user part (isdnup), Tech report GR-17-CORE.