# THE EFFECT OF PACKET LOSS ON THE RESPONSE TIMES OF WEB SERVICES

Johan Garcia, Per Hurtig, Anna Brunstrom
*Department of Computer Science, Karlstad University*
*Universitetsgatan 2, SE-651 88 Karlstad, Sweden*
*Email: {johan.garcia, per.hurtig, anna.brunstrom}@kau.se*

Keywords: Web services, performance evaluation, TCP, loss recovery, emulation

Abstract: Web services have today become an important technology for information exchange over the Internet. Although web services are designed to support interoperable machine-to-machine interaction, humans are often the final recipients of the produced information. This makes the performance of web services important from a user perspective. In this paper we present a comprehensive experimental evaluation on the response times of web services. The limited amount of data transfered in a typical web service message makes its performance sensitive to packet loss in the network and we focus our investigation on this issue. Using a web service response time model, we evaluate the performance of two typical web services over a wide range of network delays and packet loss patterns. The experiments are based on network emulation and two real protocol implementations are examined. The experimental results indicate that a single packet loss may more than double the response times of the evaluated services and lead to noticeable delays for the end user. We briefly review previous solutions that can be applied to improve performance and outline an improved approach that is based on packet loss detection at the receiver.

## 1 INTRODUCTION

The use of web services (Booth et al., 2004) is continuously expanding, not only as a means to realize distributed computing inside an organization but also as a freely available public service. More and more web sites provide services to the user that are based on the underlying use of web services. The flexibility of the web services architecture allows web services to be combined to create new end-user services. When web services are used in a context where humans are involved as final recipients of information, the response time of the web service is important as it directly affects the user experience. In this paper we present a thorough examination of the response times of web services as a function of packet loss and transport layer behaviour.

Web services use higher layer protocols such as SOAP (World Wide Web Consortium (W3C), 2003) and HTTP (Fielding et al., 1999) to transfer messages between clients and servers. On the transport layer these messages are transported by TCP (Postel, 1981). TCP provides reliable ordered delivery, and also protects the network from overload by virtue of its congestion control mechanisms. The congestion control is imperative in todays Internet as it performs the arbitration between competing flows to ensure that the flows get a reasonably fair fraction of the bandwidth while at the same time protecting the network from overload and congestion collapse. TCP's congestion control is centered around the use of packet loss as a signal that congestion is occurring. Since the congestion control uses losses as congestion markers, the congestion control also becomes intertwined with TCP's reliability mechanisms.

With regards to web services, the small data sizes in this request/response type of traffic leads to reduced efficiency for the congestion control and reliability mechanisms. Since the data sizes are so small, these mechanisms often cannot work in the most efficient manner and are to a large extent influenced by conservative default estimations of the network conditions. The effect of these efficiency problems on the web services response times are examined in this work

by means of comprehensive emulation-based experiments on real protocol implementations.

The results show that losses at the end of a connection increase the response times of web services with perceptible amounts in practically all cases, with some configurations resulting in response time increases larger than 1,7 seconds.

This paper is structured as follows. In the next section some background on web services and TCP is provided. Then follows a section describing the experimental setup and results. The next section provides a discussion of possible solutions to the problems evident in the results, and lastly the conclusions are provided.

# 2 BACKGROUND

## 2.1 Web Services Background

Web services can be seen as a means to enable distributed computing. In this respect it shares some of the goals of technologies such as CORBA (Object Management Group, 2004), RPC (Srinivasan, 1995) and Java RMI (Sun Microsystems, Inc., 2004). Web services strive to enable remote execution with a minimum of interdependency between the parties. One way of accomplishing this is by the use of open, platform neutral technologies such as HTTP, SOAP and XML (Bray et al., 2006). Although web service messages are typically exchanged between program processes without direct involvement of the users in the web service transaction, a human user is often the initiator of the action that causes the web service transfer to take place. A human user is often also the end consumer of the information resulting more or less indirectly from the web service transaction. In order to minimize the user discomfort caused by having to wait before receiving any feedback, the response times of web services are an important component.

Furthermore, the spreading of techniques such as web services mashups (Lerner, 2006) further highlights the delay characteristics. Mashups are compositions of two or more web services and are typically intended for direct end-user usage. The more web service transactions that are involved in one user interaction, the higher the risk that at least one transaction will be subject to a packet loss with a resulting undesirable increase in response time. The response time of a web service can be subdivided into smaller components. One subdivision can be made between processing delays and network delays. Processing delays are a function of the processing needed at the client and server to create messages, parse messages and do the actual execution. Network delays are caused by the delays inherent in transferring the requests and responses between the client and server. In this paper, the focus is on the network delays, and how they are affected in the presence of loss.

## 2.2 Transport Layer Background

TCP is the transport layer protocol used by web services. While the original TCP has been existing for over 30 years, it has continuously been updated, and continues to be updated, to address new challenges caused by the evolving communications technology. In the context of this study, the most relevant aspects of TCP functionality is the reliability and congestion control mechanisms as those are related to how TCP handles losses. A TCP sender has two mechanisms to detect losses, fast retransmit and timeout.

Fast retransmit (Allman et al., 1999) occurs when the sender receives three duplicate acknowledgements. The duplicate acknowledgements are sent by the receiver when it receives packets out-of-order. The reason for an out-of-order packet is either that packets have been reordered in the network, or that a packet has been lost in the network, causing all the following packets to be out-of-order. The fast retransmit threshold of three was set as a trade-off between having the sender mistakenly treat reordered packets as lost, and the delay before retransmission of a packet that has been lost.

Timeouts occur when the TCP sender has not received an acknowledgement for a certain period of time. In order to avoid having retransmissions for packets that are not lost but merely delayed in the network, the timeout value is conservatively calculated as a function of the round-trip time as measured by the the returning acknowledgments. So, for the timeout case the trade-off is between having a short enough timeout that detects losses without unnecessary delay, but not so short as to induce unnecessary retransmissions when the packet is not lost but delayed.

Out of the two described loss detection mechanisms, fast retransmit is the most desirable as it will lead to faster loss detection in practically all cases[1]. However, there are cases when fast retransmit cannot be used, and one important case in the web services context is at the end of connections. If there are too few packets to send after a loss, the receiver will not be able to generate the required number of duplicate acknowledgements. For the short connections typical in web services, this sensitive period late in the con-

[1]Fast retransmit also has a gentler congestion response than timeout, but in the present study this has practically no effect since the examined web service transfers are so short.

nection is large in relation to the overall transfer. A detailed examination of how this impacts the web services response times is provided in the next section.

# 3 EXPERIMENTAL EVALUATION

As mentioned in the previous section, packet loss plays an important role in the congestion control as well as for the reliability mechanisms. In order to study the effect of loss handling and congestion control on web services response times we have performed a comprehensive experimental campaign using real protocol implementations in an emulated environment.

## 3.1 Response Time Model

The web services response time can be divided into several components. We make a division as shown in Equation 1.

$$r_{ws} = t_{conn} + t_{req} + p_{req\_resp} + t_{resp} \quad (1)$$

The total response time $r_{ws}$ is in this model composed of the TCP connection setup ($t_{conn}$), the request transmission delay ($t_{req}$), the server-side processing delay for request parsing and response generation ($p_{req\_resp}$), and the response transmission delay ($t_{resp}$). In addition to these delays, there can also be delays at the client side to compose the request message and to parse the response message, respectively. These delays are dependent on the specific client hardware and the software implementation used in the client, and are not considered in this study.

## 3.2 Experimental Setup

The experimental setup consists of three PCs with the roles of client, server and router/emulator. The client and server communicates via the router/emulator which delays and drops packets as instructed. The experimental setup is illustrated in Figure 1. The client and server machines host programs that create requests and responses similar to web service clients and servers respectively. Since this examination has packet losses as one important variable, a specially modified version of the Dummynet emulator (Rizzo, 1997) named KauNet is used. KauNet allows precise control over packet losses, delays, and bandwidths with the possibility to specify these on a per packet basis using precomputed patterns. The ability to precisely position losses has been shown to be beneficial

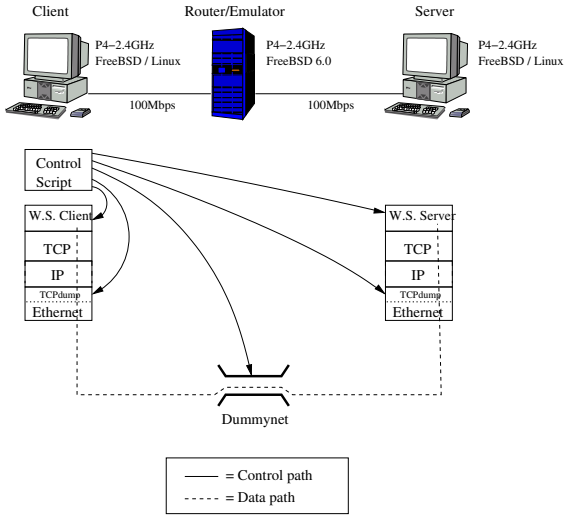


Figure 1: Physical experiment setup.

when performing protocol evaluations (Garcia et al., 2006).

To guide the configuration of the various experimental parameters, values from a study by Kim and Rosu (2004) were used. In their study they survey the length and and response times of a well known web services provider, Amazon. From the data provided by Kim and Rosu (2004) we selected two web services to use as models for our experiments. The first service was the Author service, which had a request size of 1438 bytes and a response size of 12974 bytes. The second service was the Sellerprofile which had a request size of 1283 bytes and a response size of 8031 bytes. Based on the response times reported by Kim and Rosu (2004) a $p_{req\_resp}$ of 200 ms was found to be appropriate.

The transfer of the web service requests and responses are performed by the transmission of TCP packets. Two components are used to create the emulated network delay that each packet is to be exposed to. By using two components it is possible to model packet delays that are size-dependent as well as size-independent. The size-dependent delays are mainly the transmission delays that occur at each network node. The size-dependent delays are aggregated and modeled as an effective bandwidth for the emulation. The second component of network delay is composed of delays that are independent of packet size such as propagation and queueing delays. The aggregate of the packet size independent delays, called end-to-end delay, were inserted using the delay pattern capabilities of KauNet. Each packet had a unique end-to-end delay that was drawn from a normal distribution with a mean according to the configured value, and a vari-

Table 1: Experimental Parameters.

| | | |
|---|---|---|
| Request size (bytes) | 1438 | 1283 |
| Response size (bytes) | 12974 | 8031 |
| $p_{req\_resp}$ (ms) | 200 | |
| Effective bandwidth (Kbit/s) | 160, 300, 500, 1000<br>2000, 4000, 10000 | |
| End-to-end delay (ms) | 5, 10, 20, 40<br>60, 100, 150, 200 | |

ance of 50 percent of the mean.

In order to get an indication of how issues specific to the particular TCP implementation affects the web services response times, we performed experiments with two different operating systems, FreeBSD 6.0 and Linux 2.6.15. These operating systems are commonly used in servers providing web services. A summary of the experimental parameters is provided in Table 1.

## 3.3 FreeBSD Results for Author

To examine the comprehensive results obtained from the emulation experiments, we start with a representative example of a single experimental run for FreeBSD 6.0 as shown in Figure 2. The figure shows the response time as a function of where in the connection the packet loss occurs. The y-axis shows the total web service response time according to the model above, including a $p_{req\_resp}$ of 200 ms. The x-axis represents the different loss positions possible for the data transfer in the server-to-client direction. With a response size of 12974 bytes and a maximum transfer unit (MTU) of 1500 bytes, this means that 12 packets are transfered[2] in the server-to-client direction. Loss position 0 corresponds to no packet loss and thus has the lowest response time. Similarly, a loss at position 12 also has a low response time since that packet is the final FIN-ACK required to close the connection in the opposite client-to-server direction, and a loss of this packet does not delay the transfer of data in the server-to-client direction.

The specific combination of effective bandwidth and end-to-end delay displayed in Figure 2 was chosen to provide approximately the same lossless response time as the one reported by Kim and Rosu (2004) for the case when both the client and the server were located in the US (the US-US scenario). The reported response time was 502 ms, and the experimental value was 610 ms (without losses). It can be seen that losing the packet in loss position 1 increases the response time with 3 seconds. This is expected as the timeout value used during connection establishment is set to this conservative value (Paxson and Allman, 2000). In the middle of the connection (positions 2-7) the penalty of a loss is relatively small, as the loss can be detected using the more efficient fast retransmit mechanism. Losses later in the connection (positions 8-11) result in larger increases in response times since they cannot be handled by fast retransmit, but instead have to be recovered using the slower timeout mechanism. In the specific case shown in the figure, the response time increase from having a loss late in the connection as opposed to the middle of the connection is up to 632 ms, or 103 percent[3]. The web service transactions that experience losses at the end of the connection are thus delayed for periods that are clearly large enough to be negatively perceived.

Figure 3 shows the results for a connection with the same effective bandwidth but a longer end-to-end delay to model the connection between an overseas client and a server in the US (the US-overseas scenario). The resulting response time without loss is 886 ms. The effect of losses late in the connection shown in Figure 3 is larger in absolute terms (929 ms), but smaller in relative terms (99 %) as compared to Figure 2. This of course comes from the fact that the response time with no losses becomes considerably longer when the end-to-end delay is increased.

In addition to the results shown in Figures 2 and 3, we conducted experiments for 54 other combinations of end-to-end delay and bandwidth according to values given in Table 1. To illustrate how the results vary as the end-to-end delay varies, Figure 4 shows the results for a bandwidth of 1000 Kbps, covering all the different delays. To provide an overview of all results, Figure 5 shows the relative impact of all results by calculating the percent-wise increase in response time for losses at the end of a connection. The reported value is calculated as the percent-wise increase of the mean response times of positions 8-11 over the response time for position 6.

## 3.4 Linux Results for Author

To examine the possible variations between different TCP implementations we also performed experiments using Linux. The same graphs as discussed for FreeBSD in the previous section are shown in Figures 6 to 9. These figures show that Linux has the same trend of increased response times for losses in the end of the connection that was visible for FreeBSD. This

[2]This includes the SYN-ACK packet necessary for TCP connection establishment and the final FIN-ACK packet for connection termination.

[3]This is when comparing the response times for loss positions 6 and 10.

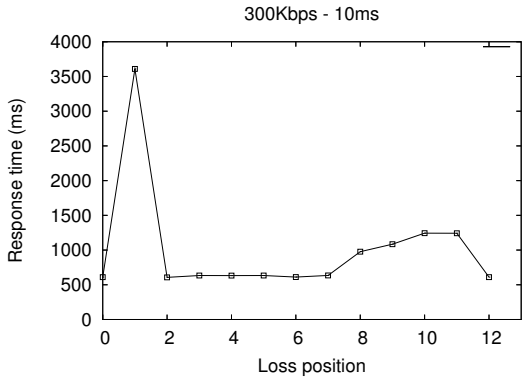


Figure 2: One FreeBSD run for US-US scenario, Author service.

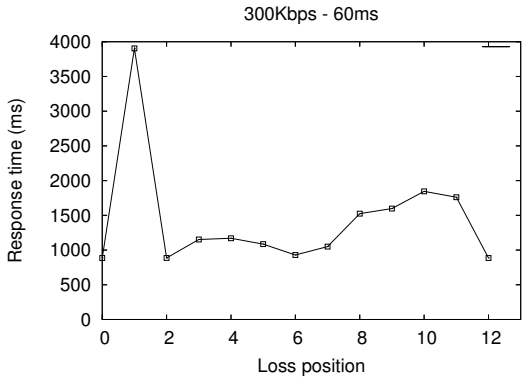


Figure 3: One FreeBSD run for US-overseas scenario, Author service.

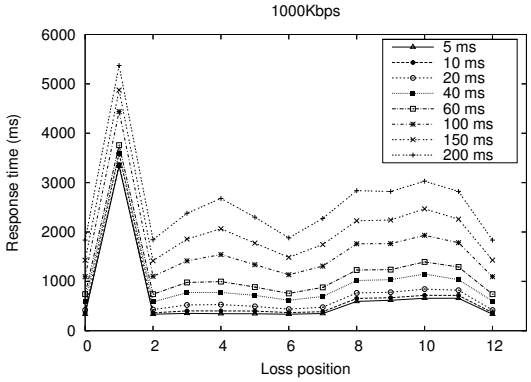


Figure 4: Impact of different delays for FreeBSD with 1000 Kbps bandwidth, Author service.

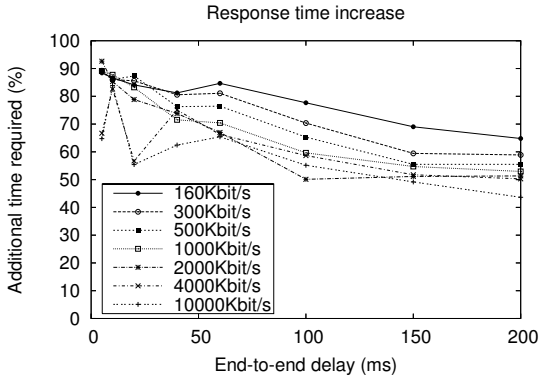


Figure 5: Free BSD response time increase for late losses, Author service.

is to be expected since the problem is inherent in the definition of TCP's reliability mechanisms. Although the trend is similar, there are some differences between the implementations. Looking at the response times without losses Linux is slower than FreeBSD for both of the end-to-end delays shown in Figures 6 and 7, with 661 ms versus 610 ms for the 10 ms end-to-end delay and 933 ms versus 886 ms for the 60 ms end-to-end delay. However, when looking at the behavior for losses at the end of the connection, it can be seen that FreeBSD actually has a sensitive region of four packets (positions 8-11) where Linux only has a sensitive region of 3 packets (positions 9-11), which clearly is beneficial for Linux [4]. Upon examination of the results shown in Figures 8 and 9 it can be seen that although Linux had slightly longer response times when there were no losses, it also had considerably less increase in the response time for the sensitive region at the end of connections. A general conclusion that can be drawn from the comparison of the two implementations is that although the behavior of the implementations differ in the details, they both share the fundamental problem of increased response times for losses at the end of connections.

[4] An examination of this issue by code inspection of the FreeBSD TCP implementation revealed that FreeBSD has a tendency to interpret one of the duplicate acknowledgements as a window update. Since window updates are not counted towards the fast retransmit threshold, FreeBSD effectively requires four duplicate acknowledgments for this case.

## 3.5 Results for Sellerprofile

In addition to the Author web service, experiments were also performed for the Sellerprofile web service. The response size used for this service was 8031 bytes. The smaller size implies that fewer packets are needed to transfer the response message. Thus, the sensitive period at the end of a connection will cover a larger fraction of the total connection length. For FreeBSD the sensitive positions are now positions

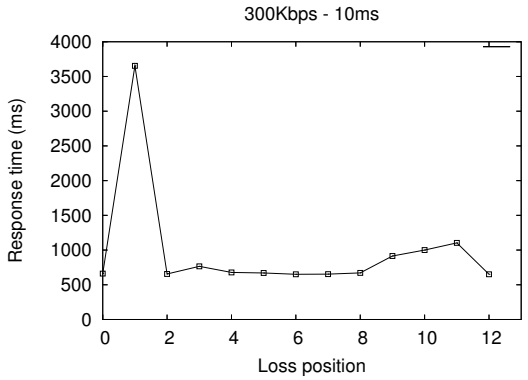


Figure 6: One Linux run for US-US scenario, Author service.

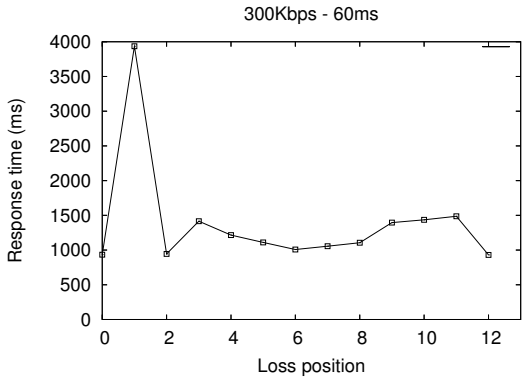


Figure 7: One Linux run for US-overseas scenario, Author service.

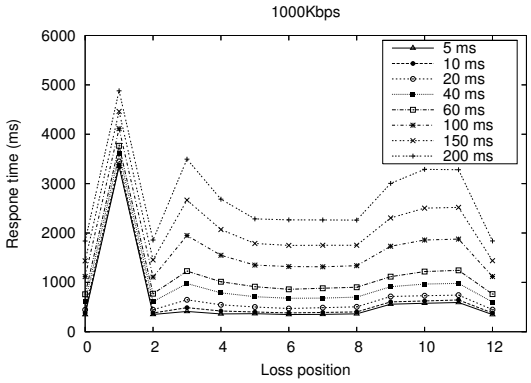


Figure 8: Impact of different end-to-end delays for Linux with 1000 Kbps bandwidth, Author service.

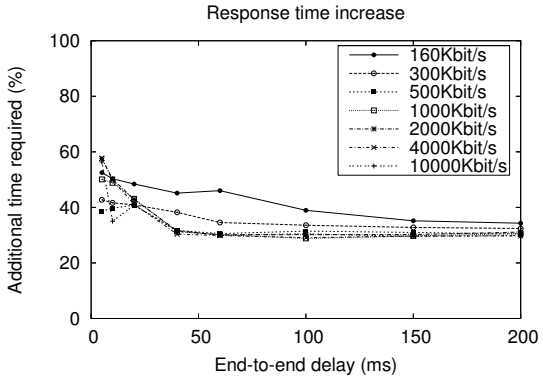


Figure 9: Linux response time increase for late losses, Author service.

5-8, and for Linux positions 6-8, as shown in Figures 10 and 12, respectively. Looking at the aggregate results shown in Figures 11 and 13, it is evident that the increases in response times are indeed larger than for the previously examined service. This provides support for the hypothesis that shorter web service transactions are more sensitive to the problem of losses late in the connection.

## 3.6 Discussion

The results highlight the difficulty of TCP's reliability mechanism to work efficiently for short flows that are typical for web services. One issue that creates a large increase in response time is the loss of the SYN-ACK[5], which will increase the response time with three seconds. However, it is hard to change this value without cross-layer knowledge of the particular network conditions before connection establishment. Since this is the first packet exchanged this timeout value must be conservatively set to allow for low bandwidth/large delay links to work with reasonable efficiency. For the other case where increases in response times were visible, i.e. losses late in the connection, there will be more knowledge about the network conditions available, which makes it easier to adapt the behavior to improve performance. Hence, the focus of this study is on losses which occur late in the connection.

For all the web service sessions that we have examined in these experiments, the response time suffered when there were losses late in the connection. The increase in response time caused by these losses were in the range of 208 ms to 1781 ms. When interpreting these results, it should be noted that they focus on the behavior in the presence of losses. In typical Internet conditions, many web services transactions will not experience any loss at all. However, for those

[5]This is also true for the SYN packet, but it goes in the direction from the client to the server which is not the focus of our investigation.

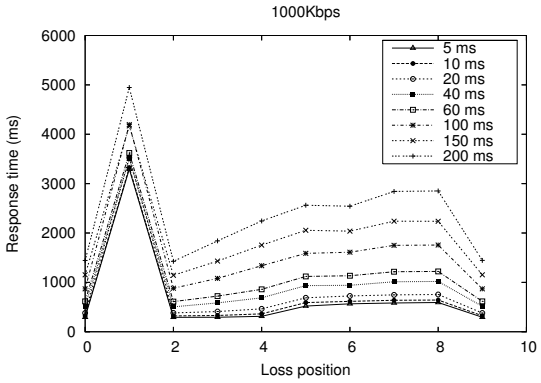


Figure 10: Impact of different end-to-end delays for FreeBSD with 1000 Kbps bandwidth, Sellerprofile service.

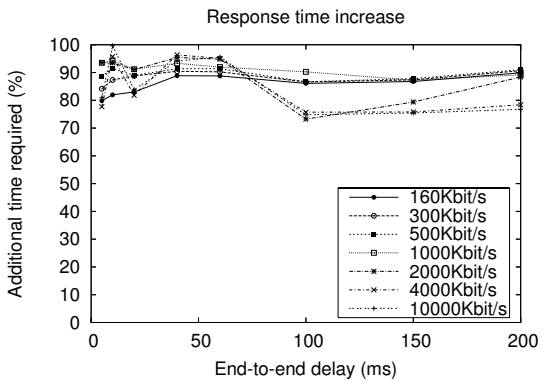


Figure 11: Response time increase for late losses, FreeBSD, Sellerprofile service.

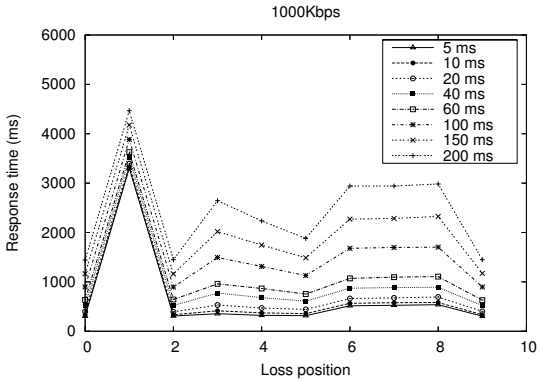


Figure 12: Impact of different end-to-end delays for Linux with 1000 Kbps bandwidth, Sellerprofile service.

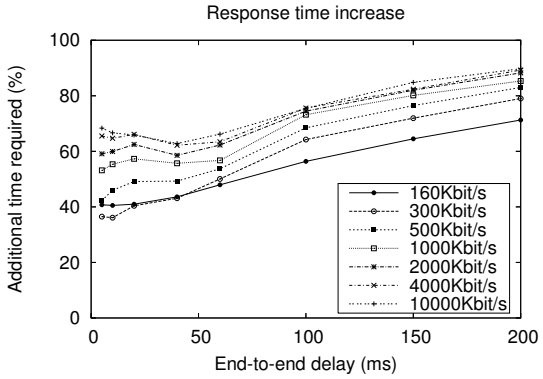


Figure 13: Response time increase for late losses, Linux, Sellerprofile service.

transactions that do experience losses, the cost in increased response time may be high. Using the Author service as an example and assuming a more typical Internet loss rate such as 2 %, on average 22 % of the web services transactions will experience losses[6]. Out of those transactions, on average 27 % will have the loss at the last three packets shown in this study to be very problematic. For shorter web services transactions, a lower percentage of the transactions will experience losses. However, for those shorter transactions that do experience losses, the risk of the loss being in the problematic region is higher. A solution that address this problem would hence be welcome.

[6]Assuming uniformly distributed losses that are independent of packet size.

# 4 POSSIBLE SOLUTIONS

As shown in the previous section there are severe consequences on the web services response times when losses occur towards the end of a connection. Although the focus of this study have been web services, the problem of losses late in the connection is also present for other client-server applications with short transfers. Various proposals have been put forth that to some extent addresses the problem. These proposal are targeting the TCP behavior, and modify it to work better for short connections that experiences packet losses. In this section these proposals are briefly described and a new hybrid scheme is outlined.

## 4.1 Modified Retransmit Calculations

One approach is to modify the calculation of the retransmission timer. This will cause the retransmission timer to expire earlier, and will decrease the un-

necessary delay until timeout at the end of a connection. However, modifications of the timeout calculation may also lead to a higher frequency of spurious timeouts, i.e causing a timeout when a packet merely is delayed and not lost. The frequency of occurrence for spurious timeouts is not well understood over the range of networks that comprise the Internet of today. There are several proposed solutions for the spurious timeout problem (for example (Sarolahti et al., 2003)) that implies that it is a common problem, but other research has shown very little frequency of spurious timeouts (Vacirca et al., 2006).

### 4.2 Smart Framing

The smart framing approach (Mellia et al., 2005) is based around the idea of splitting larger packets into smaller ones. More packets may allow better RTT estimation and an increased chance of using fast retransmit instead of timeouts. However, if the packet loss probability is independent of packet size, this scheme may lead to reduced performance. It also increases the overhead by requiring more headers and ack traffic.

### 4.3 Early Retransmit

The early retransmit approach (Allman et al., 2006) in essence entails the reduction of the duplicate acknowledgments threshold. Such a reduction will decrease the number of packets at the end of a connection that cannot be recovered by fast retransmit. Investigations in connection with this work (Allman, 2005) indicates that the reordering present in examined traces could make this solution problematic. However, other research (Jaiswal et al., 2002) indicates that networking reordering is uncommon.

### 4.4 Adaptive Duplicate Acknowledgment Generation

All of the schemes discussed above try to improve the ability of the sender to infer that a loss has occurred. This allows retransmission to occur which in turn can, in the current case, reduce the web-server response time. All these approaches have some disadvantages as already mentioned. To provide a better solution we here outline a new approach where the loss inference instead takes place at the receiver side, and the receiver then uses an implicit loss notification to make the sender perform a fast retransmission.

The first step of this scheme is to use the reception of the final FIN packet to start an adaptive duplicate acknowledgment generation module. This module checks if there is a hole in the received packet stream, i.e. a packet that is lost (or possibly reordered). It then applies a timer that is based on receiver side measurements of the packet inter-arrival times. This timer is updated when each packet is received, in contrast to the RTT estimation which typically is performed only once per congestion window. When the inter-arrival timer times out, the module sends additional dup-acks so that the total amount of dup-acks becomes three, which upon receipt at the sender will trigger a fast retransmit. This scheme has several benefits compared to the ones above; it will have a lower frequency of spurious retransmissions than the modified retransmit calculations and early retransmit schemes. Also, it will not have the always present overhead of additional headers that smart framing produces. The details of the receiver side timer calculations as well as a kernel implementation is part of planned future work.

## 5 CONCLUSIONS

We have examined the impact of packet loss on web services response times. To do this we have performed experiments on two current TCP implementations using emulation with precise loss positioning capabilities. We examined all possible loss positions for 56 combinations of effective bandwidth and end-to-end delay, for two different empirically derived request and response sizes using two different operating systems. The results show increased response times of magnitudes that are clearly perceptible to humans, and thus causes for user dissatisfaction, for many cases when the losses are placed at the end of a connection. Due to the relatively small size of many web services transactions, the sensitive area at the end is large in relation to overall transaction length. The increase in response time varied from 208 ms to 1781 ms in absolute value, and in percentage terms the increase ranged between 40 % and 112 %. Possible solutions that reduce the effect of late losses were briefly discussed, and a new receiver-based hybrid approach was sketched in order to mitigate some of the disadvantages of the other schemes. For future work we intend to focus on the development and evaluation of this hybrid approach.

## ACKNOWLEDGMENTS

The authors wish to thank Su Myeon Kim for providing the detailed measurement data on web service request and response sizes that were published in (Kim and Rosu, 2004), and Hubert Baumeister for discussions that helped to shape the content of this paper.

## REFERENCES

Allman, M. (2005). Private communication.

Allman, M., Avrachenkov, K., Ayesta, U., and Blanton, J. (2006). Early retransmit for TCP and SCTP. IETF Draft: draft-allman-tcp-early-rexmt-04.txt.

Allman, M., Paxson, V., and Stevens, W. (1999). TCP congestion control. RFC2581.

Booth, D., Haas, H., and McCabe, F. (2004). Web Services Architecture. http://www.w3.org/TR/ws-arch/.

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., and Yergeau, F. (2006). Extensible Markup Language (XML) 1.0 (Fourth Edition). W3C Recommendation.

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and Berners-Lee, T. (1999). Hypertext Transfer Protocol – HTTP/1.1. RFC2616.

Garcia, J., Alfredsson, S., and Brunstrom, A. (2006). The impact of loss generation on emulation-based protocol evaluation. In *Proc. International Conference on Parallel and Distributed Computing and Networks (PDCN 2006)*.

Jaiswal, S., Iannaccone, G., Diot, C., Kurose, J., and Towsley, D. (2002). Measurement and classification of out-of-sequence packets in a tier-1 ip backbone. In *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*, pages 113–114, New York, NY, USA. ACM Press.

Kim, S. M. and Rosu, M.-C. (2004). A survey of public web services. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers and posters (WWW 2004)*.

Lerner, R. (2006). Creating mashups. *Linux Journal*, (147).

Mellia, M., Meo, M., and Casetti, C. (2005). TCP smart framing: a segmentation algorithm to reduce TCP latency. *IEEE/ACM Transactions on Networking (TON)*, 13(2):316–329.

Object Management Group (2004). Common Object Request Broker Architecture: Core Specification. http://www.omg.org/technology/documents/corba_spec_catalog.htm.

Paxson, V. and Allman, M. (2000). Computing TCP's retransmission timer. RFC2988.

Postel, J. (1981). Transmission control protocol. RFC793.

Rizzo, L. (1997). Dummynet: a simple approach to the evaluation of network protocols. *ACM Computer Communication Review*, 27(1):31–41.

Sarolahti, P., Kojo, M., and Raatikainen, K. (2003). F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts. *SIGCOMM Computer Communications Review*, 33(2):51–63.

Srinivasan, R. (1995). RPC: Remote procedure call protocol specification version 2. RFC1831.

Sun Microsystems, Inc. (2004). Java Remote Method Invocation. http://java.sun.com/j2se/1.5.0/docs/guide/rmi/spec/rmiTOC.html.

Vacirca, F., Ziegler, T., and Hasenleithner, E. (2006). An algorithm to detect TCP spurious timeouts and its application to operational UMTS/GPRS networks. *Computer Networks*, 50(16):2981–2001.

World Wide Web Consortium (W3C) (2003). Simple object access protocol (SOAP) version 1.2. http://www.w3.org/TR/soap.