Why Is This Web Page Coming Up so Slow? Investigating the Loss of SYN Packets
(Work in Progress)

Dragana Damjanovic, Philipp Gschwandtner, and Michael Welzl

Institute of Computer Science,
University of Innsbruck, Austria
{dragana.damjanovic,philipp.gschwandtner,michael.welzl}@uibk.ac.at

Abstract. Before the first valid calculation of the round trip time for a connection, TCP sets an initial value for the retransmission timeout to 3 seconds which, in the case of the first packets getting lost, introduces a long delay. For short transfers, like web traffic, this could have a significant influence on the performance. We performed measurements to investigate how often this happens. As our measurements show, control packets (SYN and SYN/ACK packets) do get lost and delays of 3 seconds or even more (further timeouts) occur. By means of a simple example implementation, we indicate that this problem could be solved.

Keywords: network protocol, measurements, connection management.

1 Introduction

Sometimes, when browsing the Web, some page takes just too long to appear. Usually an impatient user clicks refresh one or more times and the page loads. This already happened at least once to everybody. One possibility for this behavior could be the loss of SYN or SYN/ACK packets. Web applications like browsers mainly use TCP as a transport protocol. TCP starts a connection with three-way handshaking, sending a SYN packet, waiting for a response (SYN/ACK packet) and then sending an ACK for the SYN/ACK packet which is the first packet of a TCP connection that can contain data. As recommended in [1], before the first round trip time (RTT) measurement has been done, TCP should set the retransmission timeout (RTO) to 3 seconds. Therefore in case of SYN or SYN/ACK packets getting lost TCP waits 3 seconds (or longer for further back-offs) before retransmitting them. Especially for short-lived flows, this delay can significantly degrade the performance. Reference [2] suggests that TCP may increase its initial window from one to between two and four segments. This would make TCP more robust to losses in the starting phase when the window is 1 and any loss would force a timeout, but in the case of the connection's opening packets getting lost, TCP has no choice but entering timeout.

In [3] the authors recognize this problem of lost packets belonging to the connection opening phase and their simulations show how the response time

L. Fratta et al. (Eds.): NETWORKING 2009, LNCS 5550, pp. 895–906, 2009.

can be significantly increased by just avoiding the loss of the SYN/ACK packet. They suggested the use of ECN [4] for SYN/ACK packets, but — as studies show in 2000 only 1.1% [5] of the servers were ECN-capable and till 2004 it has increased just to 2.1% [6] — ECN is not widely used. Therefore utilizing ECN for SYN/ACKs would not improve the actual performance until the number of ECN capable servers increases.

Drawn by the idea of the impact the loss of packets belonging to the connection's opening phase can have on the performance of web browsing, we investigate on how often this happens. As our measurements show this problem does exist (in average more than 0.5% of the connections experience this). Therefore we give a simple example implementation that can overcome this drawback.

In the next section we investigate related work in the area of safe connection establishment and some work done in speeding up web servers' response times. In Section 3 we present the results of our Internet measurements and in 4 we introduce and validate an example implementation that improves the performance of web browsing, followed by the conclusion of our findings.

2 Related Work

Since web traffic usually consists of very short flows (request-response connections with web clients sending a request and server answering to the request), not just the loss of packets belonging to the connection opening phase but also the whole connection opening phase itself introduces a significant delay. In the past there have been proposals for protocols more suited for transaction-oriented connections that would eliminate this delay, proposals for protocols built on top of TCP as well as protocols build from scratch.

The Stream Control Transmission protocol (SCTP) [7] is a protocol well suited for web traffic. SCTP uses a concept based on associations instead of connections, each association can have multiple streams and ordered delivery on separate streams is also supported. In the case of HTTP transfers with TCP each request is sent over a separate connection, whereas SCTP enables multiple requests to be sent over the same association. For opening a connection it uses a four-way handshake, but to reduce the delay the last two packets of the connection opening phase can carry data.

TCP for Transactions (T/TCP) [8] is a protocol suggested as an extension to the TCP protocol for distributed applications like web applications (the scenario we are investigating), that would benefit from a transaction oriented transport protocol that does not always introduce additional packet exchanges for opening and closing connections. Hosts opening a connection for the first time perform the three-way handshake and host information is cached on each side. This cache is used for validating upcoming SYN packets. Data is sent even within the first packet of a connection (SYN packet) and in case a cache entry for this host exists and the sequence number is valid (i.e. it is not a duplicate) the response to this data is sent in the next packet. In all other cases the response is sent after the three-way handshake is performed. This is an easy way to speed up short flows,

like web traffic, but it has many security drawbacks: it consumes more resources than normal TCP before a connection is open, therefore it can be attacked more easily by SYN flooding, it is more vulnerable to IP address spoofing, etc.

The Versatile Message Transaction Protocol (VMTP) [9] is another example of a transaction oriented protocol. It uses unique connection IDs for safe connection management and will fall back to three-way handshaking only in special cases (e.g. one of the hosts restarted). It is not built on top of TCP and provides better security and naming, statelessness of transactions etc. This protocol is designed for the area of distributed programming, for communication between servers and clients belonging to the same organizational entity, and it is better suited for those purposes than for an open system like the Web.

A transport protocol designed for wide area networks should implement a connection management mechanism for connection opening and closing that is immune to problems introduced by lost, duplicated or out-of-sequence packets. In the past there have been a couple of proposals for safe initialization and closing of a connection: three-way handshaking, unique connection IDs and timers. As already mentioned TCP uses a three-way handshake for opening but also for closing connections [10]. Protocols that use unique connection IDs usually need an extra mechanism for secure opening or closing, like three-way handshaking or a timer. VMTP is an example of this kind of protocol and as mentioned before it uses a three-way handshaking in some cases. The Delta-t transport protocol [11] uses a timer-based connection management. Connections are handled without any connection management packet exchanges. Communication is established between ports of processes and Delta-t defines a stream as the unique triple (destination port, source port, stream number), where ports are identified by 64 bit addresses unique for the whole network. The Delta-t connection management is built on the idea that there are permanent error- and flow- controlled connections between all possible streams. The time-based mechanism automatically recognizes whether a connection is in the default state and accordingly allocates or deallocates the connection state without any packet exchange. Delta-t and VMTP are just two examples of protocols that deal with connection opening without any packet exchange, but none of these protocols are used in the Internet today.

3 Measurements

To investigate how often the stated problem occurs we observed web traffic. Here we only consider the relevant part for our topic, which is the connection opening phase. The loss of connection opening packets can be detected on the client side and on the server side. There are 4 possible cases that can be distinguished:

client side

duplicate SYN: If a second duplicate SYN is sent by the client, either the first SYN or the corresponding SYN/ACK must have been lost.

duplicate SYN/ACK: If a duplicate SYN/ACK is received, we can safely deduce that the corresponding ACK has been lost.

server side

duplicate SYN: If a second duplicate SYN packet is received by the server, the client did not receive the corresponding SYN/ACK.

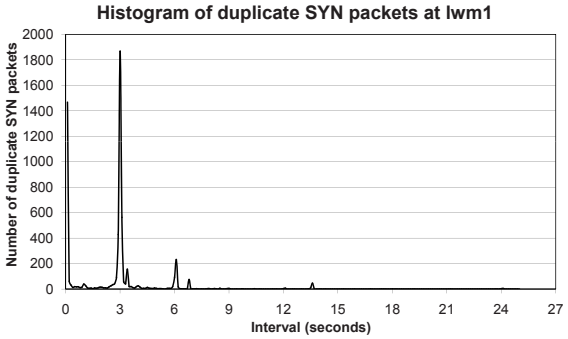
duplicate SYN/ACK: If a second duplicate SYN/ACK is sent, either the first SYN/ACK or the corresponding ACK was lost.

Measurements were taken on the server side as well as on the client side, using tcpdump and subsequent analysis of the trace files. For every packet, the TCP header is inspected to determine whether it is an SYN, SYN/ACK or ACK responding to the received SYN/ACK and therefore a part of the three-way handshake of TCP. Then, the analyzer checks for duplicate SYN or SYN/ACK packets (packets that hold the same source and destination IP address and port as well as the same sequence number) to detect if packets belonging to the connection opening phase got lost. We also measured the time difference between such repeated packets. If this time difference matches the standard initial RTO time (3 seconds or further back-offs with the RTO multiplied by 2 each time) we can safely assume for the majority of the cases that the duplicate packet was automatically transmitted. For all the other cases, other factors have to be taken into account.

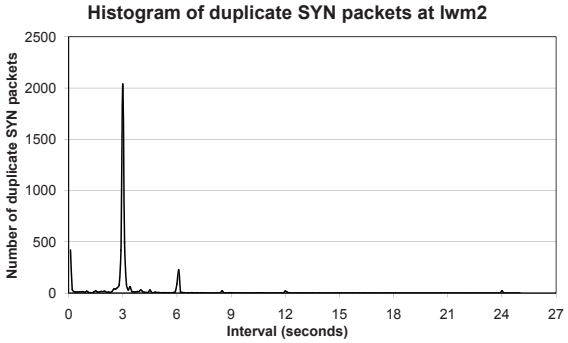
3.1 Web Mail Server (Server)

The University of Innsbruck provides two separate Linux-based servers for web mail that we used for our server-side measurements, one for the employees (http://web-mail1.uibk.ac.at) and a second one for the students (http://web-mail2.uibk.ac.at). We recorded roughly 208.000.000 packets for a duration of 12 days. The following results were obtained by doing server-side measurements at the former. A total of 737.188 requests for initiating a connection have been logged, 730.523 of which were successful (99,1%). 5162 (0.7%) of those successful connections required more than one SYN packet, which shows a real occurrence of lost SYN packets. Figure 1(a) shows the results of this measurement as a histogram, illustrating the time interval between transmitting duplicate SYN packets. There are peaks of duplicate SYNs at 3 and 6 seconds, which match the first two time intervals for the RTO timer. The fact that there are many SYN packets that approximately fit into a time interval but do not do so precisely can be explained with variances in the calculation of the initial RTO [1]. Therefore we can assume that those duplicate SYN requests were automatically generated.

Furthermore there are smaller peaks at the slightly higher time values 3.4 seconds, 6.8 seconds and 13.6 seconds, which also suggests an automated generation with just a higher starting value or increasing delay in network. Another peak is visible at approximately 0-0.3 seconds. The duplicate SYN packets arriving at those small intervals (and also at any other time interval than the standard ones used by RTO) might occur due to potentially faulty TCP implementations, deliberately changed RTO settings or possibly other factors like lower-level frame collisions causing switches to forward a duplicate of a frame. However, the phenomenon of those short interval packets cannot be explained by impatient users



(a) Webmail Server lwm1



(b) Webmail Server lwm2

Fig. 1. Webmail Servers lwm1 and lwm2

clicking on "refresh" or accidental double-clicks, since it is not possible to force the standard TCP implementations to resend a second SYN containing the same source IP and port. We investigated further on this topic but found no way at the application layer of causing TCP to resend a duplicate SYN. The reaction of impatient users still solves the problem however, since today's web browsers simply open a new TCP connection for each refresh request.

The second measurement at the student's web mail server showed similar results. Again, the highest peaks are positioned at the standard initial RTO interval times and smaller peaks at slightly increased timings as before. Also quite a number of duplicate SYNs were received during non-automatic interval times, suggesting either miscalculation or similar phenomena as described above (mainly at 0 to 6 seconds, but continuously small occurrences up until 18 seconds). The results are illustrated in Figure 1(b).

3.2 Internet Proxy (Client)

Moreover we recorded traffic at the university's Internet proxy using the client-side measurement method. For a duration of two days we recorded 154.000.000

packets. A total of 1.721.382 connection attempts have been logged, 1.708.442 of which were successful (99,2%). Multiple SYN/ACKs (and SYNs) were necessary in 2.148 of those successful connections. Figure 2(a) illustrates the interval between the original and duplicate SYN/ACKs. First of all there are quite less SYN/ACKs than SYN packets compared to our previous measurements, which can be explained by the nature of the client-side measurement. The server sends a duplicate SYN/ACK only if the first SYN/ACK or its corresponding ACK got lost. If the former is the case, the client counts only one SYN/ACK - a phenomenon that also occurred in our other measurements - which results in less SYN/ACKs being registered on the client side. However this is not the case for measuring SYN packets, since we notice every transmission of SYNs on the client side, regardless of their success. Furthermore the figure illustrates a rather high number of duplicate SYN/ACKs at 0 to 4 seconds, which might be a result of high server load and therefore delayed response times. Additionally there are peaks at 1.5 seconds, 9 seconds and 16.5 seconds, which do not match any of the suggested RTO intervals. Hence we assume that these peaks are caused by miscalculation of the RTO or non-compliance of the specification.

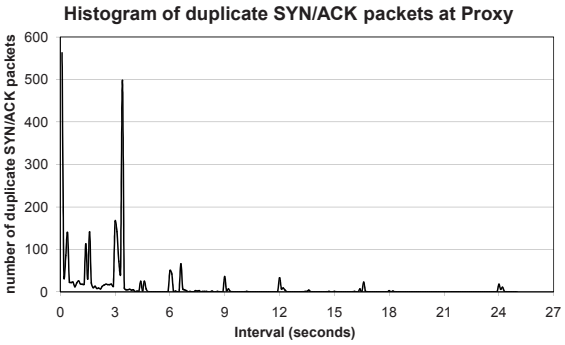
3.3 Free Tracelogs of LBNL/ICSI (Server and Client)

In addition to the tests we performed at the servers of our university, we examined the logfiles of the LBNL/ICSI Enterprise Tracing Project, which contain both server- and client-side measurement data that is freely available for download.[1] The files contain three months of measurement data consisting of approximately 5.000.000 packets involving HTTP. The data that is relevant to our topic includes 232.852 connection attempts, 219.867 of which were successful (94.4%). For 4103 (1.9%) of those successful connections multiple SYNs were necessary. The results of duplicate SYNs are illustrated in Figure 2(b) and show that, compared to our own measurements, there are much less duplicate SYNs at non-automatic retransmission time intervals. This can be explained by different measuring conditions, since the data downloaded from LBNL/ICSI was retrieved in a local area network, whereas our measurements most probably include everyday technologies like ADSL, cable Internet and WLAN. Since they are likely to be more error-prone than wired Ethernet connections, it is deductible that they show more irregularities due to lost packets. Apart from the peaks at the standard RTO interval times, there are peaks at 0.5 and 20 seconds. Since they are too protruding to be random deviations it might be possible that some specific program or deliberately changed RTO setting is responsible for those peaks.

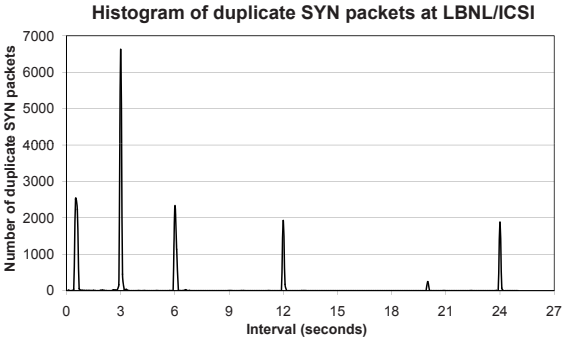
3.4 Summary of the Measurements

Additionally to the measurements described thoroughly above, we took measurements at Alupress AG[2], a company operating in the field of metal processing,

[1] http://www.icir.org/enterprise-tracing/download.html
[2] http://www.alupress.net



(a) Internet Proxy Server



(b) LBNL/ICSI Tracefiles

Fig. 2. Measurements taken at the Internet Proxy Server of UIBK and LBNL/ICSI

that showed similar results and are therefore not described in detail. In all our measurements (as illustrated by Table 1) we found that lost packets belonging to the three-way handshake occur at a relatively high rate of 0.5%. Furthermore we deduce from the logfiles of our own measurements that the phenomenon of duplicate SYN packets arriving only 0 to 0.3 seconds after the original SYN

Table 1. Summary of all measurements

	UIBK Webmail LWM1	UIBK Webmail LWM2	Alupress Proxy	UIBK Proxy	LBNL/ ICSI
Connection tries total	737.188	665.829	75.493	1.721.382	232.852
Connection tries successful	730.523	659.913	75.049	1.708.442	219.867
Connection tries not successful	6.665	5.916	444	12.940	12.985
multiple SYNs	5.968	4.838	480	19.452	22.268
multiple SYNs for successful conn.	5.162	4.064	214	5.696	4.103
multiple SYNs for unsuccessful conn.	806	774	266	13.756	18.165
multiple SYNs outside standard interval	690	577	48	23	428
SYNs per successful connection	1,007	1,006	1,003	1,003	1,019
duplicate SYN every X connection	142	162	351	300	54

packet seems to be confined to a relatively small number of source hosts. They might use very small RTO settings that match those time intervals or otherwise modified TCP implementations that deviate from the suggested standard behaviour. Since one of the requirements of ZID - our university's central information technology service, who gave us permission to record and analyse traffic data - was the anonymization of host IP addresses, we cannot investigate further on the origin of those particular SYN packets.

4 Enhancing Performance

There are different ways the performance in case of lost control packets could be improved. A simple solution to enhance the performance is to change the TCP settings in the operating system, e.g. setting the initial retransmission value to something smaller than 3 seconds, but this will then apply for every TCP connection and possibly introduce unnecessary retransmissions and could even cause TCP to fail in certain cases of extreme delay. Therefore the Internet standard recommends a rather conservative value. For example setting the value to 100 ms would create problems for establishing connections with a longer RTT. Since two SYN packets have a higher probability to arrive at their destination, a possibility would be to send each SYN packet twice with a short time interval between. This would reduce the probability of failures at the cost of a little bit more traffic in the network. A better solution, that would reduce traffic, would be to perform some statistical evaluation on how long it normally takes to get an ACK for a SYN packet and to set the retransmission timeout accordingly. Here we present an example implementation that shows that performance can be improved. We implemented a small tool (*syn_optimizer*) that runs at the application layer. The tool keeps a copy of sent packets belonging to the connection opening phase and in case the corresponding acknowledgement does not arrive, it retransmits the packet. The tool can be applied just for a certain port (in this case port 80) and the delay before retransmission of a non-acknowledgement packet is also configurable.

The loss of packets belonging to TCP's initialization can be recognized on the client as well as on the server side and the problem can be solved on both sides. As already mentioned on the client side a loss of a SYN or SYN/ACK packet can be noticed, on the other hand, the server side cannot recognize the loss of a SYN packet. Therefore the server side implementation has some drawbacks.

4.1 Client Side

On the client side the tool monitors sent packets and keeps a copy of sent SYN packets. In case a corresponding SYN/ACK packet has not been received after a certain delay (SYN packet or SYN/ACK packet has been lost) the SYN packet is retransmitted. The delay and number of retransmissions can be set via an input parameter. Furthermore the tool will only retransmit packets with a certain destination port (for web traffic: port 80). The delay parameter should not be set too low because it can cause unnecessary retransmissions on a connection

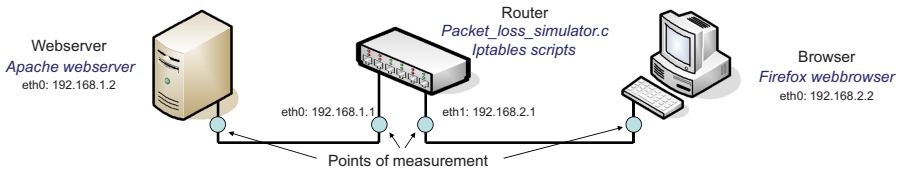


Fig. 3. Illustration of our testbed

with a longer RTT. In the case of this tool being installed at the net edge and then applying the changes for all traffic the choice of the delay should be even more conservative. The tool can be called like in the following example:

$syn\_optimizer - d\ 200\ - r\ 3\ - c\ - p\ 80$,

where d denotes the delay, r represents the number of retransmission, c indicates it is the client side (s for the server side) and p stands for port.

We tested the tool using a small testbed consisting of three computers, as illustrated in Figure 3. All computers run Linux (kernel version 2.6). One of the computers is used as a router and has two network cards (Eth0: Intel PRO/1000 and Eth1: Intel PRO/100) The other two computers are used as a web server (using the Apache Web Server, with Eth0: Intel PRO/100) and as a client (using Firefox, with Eth0: Broadcom Tigon 3 Gigabit).

To test *syn_optimizer* we simulated the packet loss on the router. We ran a couple of tests: without any packet loss, with one SYN packet lost, with one SYN/ACK packet lost, with both SYN and SYN/ACK packets lost and finally with 3 SYN packets lost. In each case the waiting time before *syn_optimizer* would retransmit a packet was set to 200 ms and the number of retransmissions was set to 3. Using the Wireshark[3] tool we observed packets sent on the link between the client and the router as well as packets sent between the router and the server.

Table 2. The test with 1 SYN packet lost; without *syn_optimizer*

Packets sent from the client to the router:					
No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.2	192.168.2.2	TCP	40877 > http [SYN] Seq=0 Win=
2	2.996952	192.168.1.2	192.168.2.2	TCP	40877 > http [SYN] Seq=0 Win=
3	3.000432	192.168.2.2	192.168.1.2	TCP	http > 40877 [SYN, ACK] Seq=0
4	3.000539	192.168.1.2	192.168.2.2	TCP	40877 > http [ACK] Seq=1 Ack=
5	3.000689	192.168.1.2	192.168.2.2	HTTP	GET / HTTP/1.1
6	3.000864	192.168.2.2	192.168.1.2	TCP	http > 40877 [ACK] Seq=1 Ack=
7	3.001320	192.168.2.2	192.168.1.2	HTTP	HTTP/1.1 304 Not Modified
8	3.001437	192.168.1.2	192.168.2.2	TCP	40877 > http [ACK] Seq=552 Ac
9	8.001572	192.168.2.2	192.168.1.2	TCP	http > 40877 [FIN, ACK] Seq=18

Packets sent from the router to the server:					
No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.2	192.168.2.2	TCP	40877 > http [SYN] Seq=0 Win=
2	0.000119	192.168.2.2	192.168.1.2	TCP	http > 40877 [SYN, ACK] Seq=0
3	0.000279	192.168.1.2	192.168.2.2	TCP	40877 > http [ACK] Seq=1 Ack=
4	0.000433	192.168.1.2	192.168.2.2	HTTP	GET / HTTP/1.1
5	0.000587	192.168.2.2	192.168.1.2	TCP	http > 40877 [ACK] Seq=1 Ack=
6	0.001037	192.168.2.2	192.168.1.2	HTTP	HTTP/1.1 304 Not Modified
7	0.001182	192.168.1.2	192.168.2.2	TCP	40877 > http [ACK] Seq=552 Ac
8	5.001267	192.168.2.2	192.168.1.2	TCP	http > 40877 [FIN, ACK] Seq=1

[3] http://www.wireshark.org/

Table 3. The test with 1 SYN packet lost; with *syn_optimizer*

Packets sent from the client to the router:					
No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.2	192.168.2.2	TCP	57993 > http [SYN] Seq=0 Win=
2	0.258130	192.168.1.2	192.168.2.2	TCP	57993 > http [SYN] Seq=0 Win=
3	0.259776	192.168.2.2	192.168.1.2	TCP	http > 57993 [SYN, ACK] Seq=0
4	0.259903	192.168.1.2	192.168.2.2	TCP	57993 > http [ACK] Seq=1 Ack=
5	0.260003	192.168.1.2	192.168.2.2	HTTP	GET / HTTP/1.1
6	0.260174	192.168.2.2	192.168.1.2	TCP	http > 57993 [ACK] Seq=1 Ack=
7	0.260630	192.168.2.2	192.168.1.2	HTTP	HTTP/1.1 304 Not Modified
8	0.260751	192.168.1.2	192.168.2.2	TCP	57993 > http [ACK] Seq=552 Ac
9	5.258758	192.168.2.2	192.168.1.2	TCP	http > 57993 [FIN, ACK] Seq=1
Packets sent from the router to the server:					
No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.1.2	192.168.2.2	TCP	57993 > http [SYN] Seq=0 Win=
2	0.000135	192.168.2.2	192.168.1.2	TCP	http > 57993 [SYN, ACK] Seq=0
3	0.000319	192.168.1.2	192.168.2.2	TCP	57993 > http [ACK] Seq=1 Ack=
4	0.000418	192.168.1.2	192.168.2.2	HTTP	GET / HTTP/1.1
5	0.000571	192.168.2.2	192.168.1.2	TCP	http > 57993 [ACK] Seq=1 Ack=
6	0.001008	192.168.2.2	192.168.1.2	HTTP	HTTP/1.1 304 Not Modified
7	0.001163	192.168.1.2	192.168.2.2	TCP	57993 > http [ACK] Seq=552 Ac
8	4.999125	192.168.2.2	192.168.1.2	TCP	http > 57993 [FIN, ACK] Seq=1

Table 2 shows packets seen on the link between the Web server and the router and on the link between the router and the client. In this case the first SYN packet was lost and the tool was not applied. As can be seen the first retransmission of the lost SYN packet appeared after almost 3 seconds and an HTTP GET command was sent after 3.00 seconds. The results when *syn_optimizer* was applied are shown in Table 3. We set a delay of 200 ms for resending the SYN packet and the SYN packet was retransmitted after about 258ms. A possible reason for this extra delay of about 60 ms is that our tool runs on the application level. The HTTP GET command was sent after 260 ms which was 2.74 seconds faster than without our tool. Some test showed that the response can be even 20.29 seconds faster with the tool. The test with the loss of 3 SYN packets showed that without our tool applied the HTTP GET was sent after 21.02 seconds compared to just 0.73 seconds with our tool in use. The summary of all tests is shown in Table 4.

4.2 Server Side

As we already mentioned a performance optimization in case of SYN/ACK packet loss can be done on the server side too. In this case our tool will monitor sent packets and capture SYN/ACK packets. If the corresponding acknowledgement for a SYN/ACK packet does not arrive after a certain delay, our tool will retransmit the SYN/ACK packet.

Table 4. Improvement using *syn_optimizer* on the client side

number of lost SYN packets	number of lost SYN/ACK packets	without *syn_optimizer*	with *syn_optimizer*	difference
1	0	3.00	0.26	2.74
0	1	3.00	0.30	2.70
1	1	6.50	0.45	6.05
3	0	21.02	0.73	20.29

We used the same test setup as in section 4.1 and observed the behavior of our tool in following cases: 1 SYN packet was lost, 1 SYN/ACK packet was lost, both SYN and SYN/ACK packets were lost and 3 SYN/ACK packets were lost.

In the case of the loss of a SYN packet our tool did not show any improvement, because the loss of a SYN packet cannot be detected on the server side. Since the results for this scenario are similar we will discuss one in detail. The client retransmitted the lost SYN packet after the TCP timeout expired (after 3 seconds). Since the next two SYN/ACK packets were lost too, the TCP timeout triggered two more times: once at the server (at 3.800 seconds) and once more at the client (at 8.99 seconds, the RTO was doubled). With the optimizer in place, at 0.0 seconds the server received a SYN packet and immediately sent a SYN/ACK packet. This packet was saved by the tool and since no acknowledgement was received in the next 200 ms, *syn_optimizer* resent the SYN/ACK packet at 0.316 seconds. Because of not receiving any acknowledgments in next two 200 ms intervals the SYN/ACK packet was retransmitted again at 0.560 ms and 0.801 ms. After the third retransmission the acknowledgment was received. The HTTP GET command was sent after 0.8 seconds which was 8.2 seconds faster than without *syn_optimizer* (in that case the time interval was 9.0 seconds). The summary of our other tests can be seen below:

number of lost SYN packets	number of lost SYN/ACK packets	without *syn_optimizer*	with *syn_optimizer*	difference
1	0	2.99	3.00	-0.01
0	1	3.00	0.33	2.67
1	1	6.27	3.28	2.99
0	3	9.00	0.80	8.20

As is evident using this simple tool can increase the performance of web browsing. By using it on the client side, the performance can be improved in all cases. On the server side the tool cannot detect the loss of SYN packets, but in any case congestion is more likely to happen on the way from the server to clients rather than in the opposite direction. Still, installing such a tool at the server will improve performance of all users connecting to the server.

5 Conclusion

The results of our measurements show that the loss of SYN packets is occurring at a relatively high rate of 0.5%, forcing the host to resend duplicate SYN packets. While some of those duplicate packets are sent after time intervals that do not match the suggested standards, the majority of them are retransmitted at the standard RTO interval times, which can significantly delay short lived data transfers. To overcome this problem we investigated the possibility of reducing the RTO time interval, which has the disadvantage of possibly introducing unnecessary retransmissions for connections with higher round-trip times. Therefore we show, by means of a simple example implementation, that the performance can be improved. Our tool keeps copies of sent control packets and retransmits

them in case no response has been received for a certain amount of time. It can be configured only to operate on a certain port and for a specific timeout. The results show that data transfer time (in our tests an HTTP GET command) can be improved by between 2.74 and 20.29 seconds. Similar improvements can also be observed when applying the tool on the server side for the retransmission of SYN/ACK packets.

Acknowledgements

We would like to thank both Walter Müller and Benjamin Kaser, who assisted us in gathering the measurement data for this paper.

References

1. Paxson, V., Allman, M.: Computing TCP's retransmission timer. RFC 2988, Internet Engineering Task Force (November 2000)
2. Allman, M., Floyd, S., Partridge, C.: Increasing TCP's Initial Window. RFC 2414 (Experimental), Obsoleted by RFC 3390 (September 1998)
3. Kuzmanovic, A.: The power of Explicit Congestion Notification. In: SIGCOMM 2005: Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications, pp. 61–72. ACM, New York (2005)
4. Ramakrishnan, K., Floyd, S., Black, D.: The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168 (Proposed Standard) (September 2001)
5. Padhye, J., Floyd, S.: Identifying the (TCP) Behavior of Web Servers. In: ACM SIGCOMM (2001)
6. Medina, A., Allman, M., Floyd, S.: Measuring the Evolution of Transport Protocols in the Internet. SIGCOMM Comput. Commun. Rev. 35, 37–52 (2005)
7. Stewart, R.: Stream Control Transmission Protocol. RFC 4960 (Proposed Standard) (September 2007)
8. Braden, R.: T/TCP – TCP Extensions for Transactions Functional Specification. RFC 1644 (Experimental) (July 1994)
9. Cheriton, D.: VMTP: a transport protocol for the next generation of communication systems. SIGCOMM Comput. Commun. Rev. 16(3), 406–415 (1986)
10. Postel, J.: Transmission Control Protocol. RFC 793 (Standard), Updated by RFC 3168 (September 1981)
11. Watson, R.W.: The Delta-t transport protocol: Features and experience. In: Proc. IEEE 14th Conf. Local Comput. Networks, Minneapolis, MN, October 1989, pp. 399–407 (1989)