FUNCTIONALITY OF TCP IMPLEMENTATIONS INSTALLED ON THE CARNET FESB NODE

JULIJE OŽEGOVIĆ, MILUTIN KAPOV

FESB Split, tel. 021 563777, fax. 563877
Julije.Ozegovic@fesb.hr, Milutin.Kapov@fesb.hr

Abstract: The TCP over slow SLIP line measurement traces of purposely initiated FTP sessions are analyzed in the context of present standardization. Results are used to tune the TCP parameters where possible. Repeated measurements show improvements in the line utilization. However, further TCP mechanism development is needed to achieve optimal results.

KEYWORDS: Internet, measurement, TCP, serial line, congestion avoidance

INTRODUCTION

Series of measurements were performed to locate and eliminate the causes of poor serial line performance. Measurements and performance results are presented in [11]. Serial Line Internet Protocol (SLIP, [14]) utilization varies, depending on line rate and Maximum Segment Size (MSS), from extremely low 18% to very good 95%. Recorded traces show differences in Transmission Control Protocol (TCP) implementations between various UNIX brands. The performance depends not only on initial Retransmission Timeout (RTO), but also on TCP algorithms' variations.

In this work, the TCP functionality analysis by algorithm check is described. In Chapter 1. a survey of the current TCP standardization is given. Chapter 2. contains the analysis of measurement traces. In Chapter 3. the TCP parameter tuning is proposed and checked. Overall TCP algorithm analysis is outlined in Chapter 4. Conclusions are derived.

1. CURRENT TCP STANDARDIZATION

The basic TCP standard can be found in [12]. However, since it suffered several functional deficiencies and ambiguities, the efforts to upgrade it have arisen soon. In [3] one can find the proposal for "Silly Window Syndrome" avoidance and "Delayed ACK" algorithms, [13] clarifies the issue of MSS, [10] deals with the congestion control, modifying transmitter and receiver behaviour and proposing ICMP Source Quench message use. Several papers appeared in Proceedings and Journals, including [9] dealing with Retransmission Timeout (RTO) algorithms, and [4] where Slow Start, Congestion Avoidance and Round Trip Time (RTT) measurement algorithms are proposed. Finally, in [1] some of the proposed modifications are accepted as official standards, clarifying ambiguities and proposing required as well as recommended host characteristics.

The protocol standardization includes Protocol Data Unit (PDU) specification, which can be stated as “exterior” protocol definition, because communicating processes know about each other only by means of PDUs received. However, it is widely understood that PDU definition is not sufficient. Protocol standardization includes various algorithm specifications, which can be stated as “interior” protocol definitions, determining the behaviour of PDU senders and receivers. The main organizational difference between the two is that updates of exterior specification need simultaneous software changes in all network computers (and nodes), while interior protocol changes may provide compatibility with previous implementations. No simultaneous software changes are required.

In TCP standardization, the PDU header format as defined in [12] has not been changed afterwards. Several proposals for various option additions have been issued, but none of them officially accepted. Main TCP extensions were specified as interior definitions. Software developers must include the algorithms proposed in [3, 4, 9, 10 and 13]. Namely, TCP implementation should avoid silly windows, delay ACKs, inhibit sending small packets (Neagle’s algorithm), use exponential backoff of RTO, hold backoff RTO after first acknowledgment (Karn’s algorithm), calculate RTO from RTT using variance, use slow start algorithm, and use congestion avoidance algorithm with congestion window linearly increasing and exponentially decreasing.

After [1], work on further TCP refinement has been concentrated in the field of congestion control and avoidance. Several interior definitions have been proposed, but no one officially accepted. The algorithms under development include fast retransmit [4], fast recovery [5], selective acknowledgments [7], and congestion avoidance on the basis of RTT measurements [2, 16]. Current TCP implementations include fast retransmit and receiver sending ACK on every segment algorithms (known as Tahoe TCP). Some vendors deliver TCP with fast recovery included (known as Reno TCP), and some implement high speed - high delay network extensions [8].

2. ANALYSIS OF TCP TRACES

Measurements of serial line utilization were performed systematically as described in [11]. Traces from that activity were used to analyze the functionality of TCP implementations available. Recorded traces were captured by TCPDUMP utility [6] on LINUX host, containing TCP headers of intentionally activated FTP sessions. A 38 kbyte compressed file was transferred.

The measurement network topology is shown in Figure 2.1. Data were transferred from TCP sender over Ethernet installation to the serial line controller Cisco CS500, then via slow SLIP connection to the TCP receiver. The impact of slow line on TCP connections in this topology is dominant and other delays can be neglected. This can be quantified in terms of Minimal RTT (RTT_0, [11]). Chosen topology simulates actual modem access connections well, considering average users run mostly Web browsers through the local proxy server.

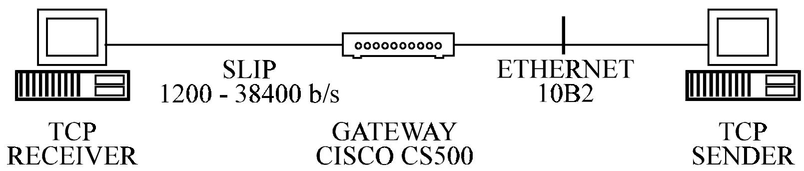

Figure 2.1. - The measurement network topology

Three brands of TCP implementations were available. There were SUN SPARCstation 20 workstation with Solaris 2.4 rev. 3/95 operating system software installed, DECstation 5000 workstation running ULTRIX V4.2a rev47 OS and LINUX Slackware 1.2.3. systems on DEC Venturis PCs. In all experiments, the TCP receiver was Trumpet Winsock V2.0 revB with receive window set to 10*MSS. Typical and worst case behaviours are analyzed. Selected traces are not comparable. For performance comparison, see [11].

2.1. TCP traces of Solaris adria.fesb.hr

The Solaris acceptably good trace is observed for rate R=14400 b/s and MSS=512 bytes. The typically inadequate trace is for R=14400 b/s and MSS=1460 bytes, Figure 2.2.

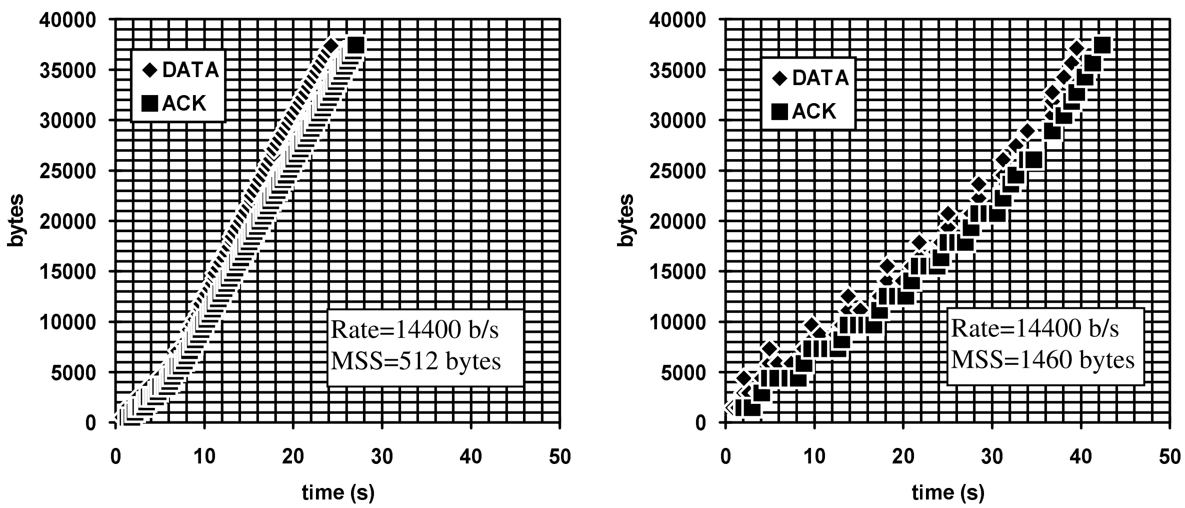

Figure 2.2. - The Solaris adria.fesb.hr traces

With MSS=512, Retransmission Timeout Counter (RTO) stabilizes after 6s (5000 data bytes) and data transfer proceeds with theoretical maximum speed. Setting MSS=1460, RTO never stabilizes, and total transfer time is increased by 15s. The TCP behaviour is analyzed on the magnified chart of R=9600 b/s and MSS=1460 bytes, Figure 2.3.

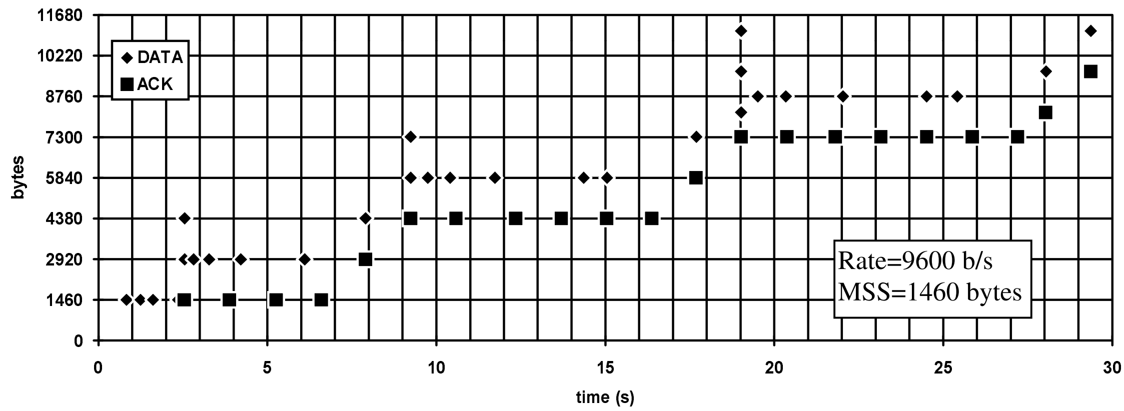

Figure 2.3. - The magnified Solaris adria.fesb.hr trace

The first segment is transmitted at time t=0.9s, and suddenly retransmitted four more times because of inadequately tuned initial RTO. First acknowledgment at t=2.6s increases congestion window as specified for slow start algorithm. Two more segments are transmitted. However, segment 2920 is unexpectedly retransmitted four more times because the RTO backoff started from the beginning. Solaris TCP simply does not keep backoff time (Karn's algorithm) as proposed in [1, 9]. This is the main reason for prolonged instability of the connection. ACK received at t=7.9s reinitiates slow start mechanism as expected [4], but the same malfunctioning pattern can be observed again and again. More adequate initial RTO/RTT ratio leads eventually to RTO stabilization as seen in Figure 2.2. left. Fast retransmission takes place at t=15s and again at t=24.5s. It is interesting that RTO retransmissions occur concurrently with fast retransmissions. A segment packetization glitch can be observed at t=19s.

2.2. TCP traces of ULTRIX papundek.fesb.hr

The ULTRIX very good trace is for R=14400 b/s and MSS=1460 bytes. The typically bad trace is for R=14400 b/s and MSS=1006 bytes, Figure 2.4. One can expect better results with lower MSS (higher initial RTO/RTT ratio); however, ULTRIX TCP did succeed occasionally to show traces similar to Figure 2.4. left.

In the left graph of Figure 2.4. we can see data transfer performing near the maximum theoretical speed, with no retransmissions.. On the right, several retransmissions are observed and total transfer time is increased by 7s. Connection stabilizes at t=17s after almost 14k bytes of data had been transferred.

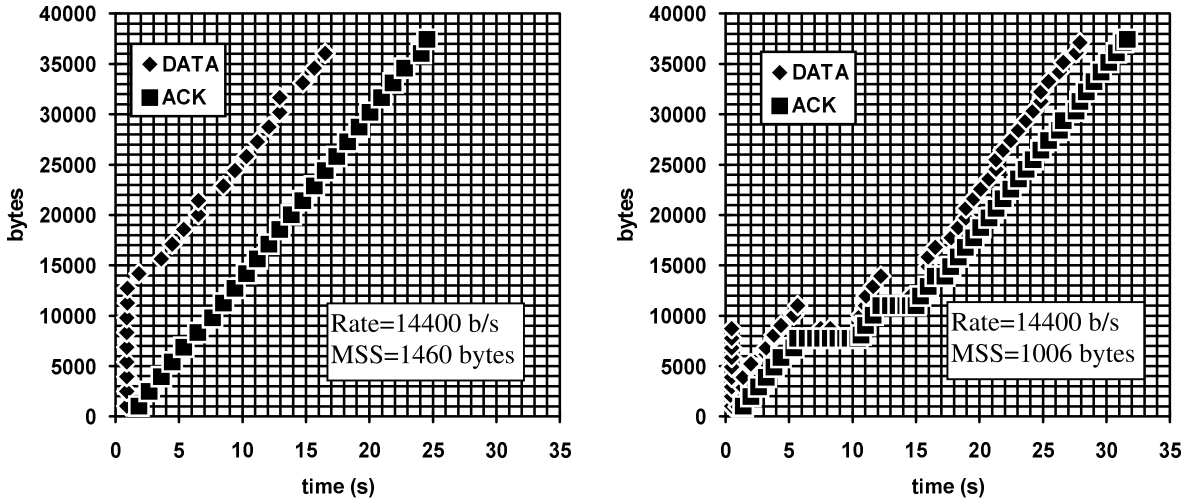

Figure 2.4. - The ULTRIX papundek.fesb.hr traces

The TCP behaviour is analyzed on the magnified chart of R=14400 b/s and MSS=1006 bytes, Figure 2.5.

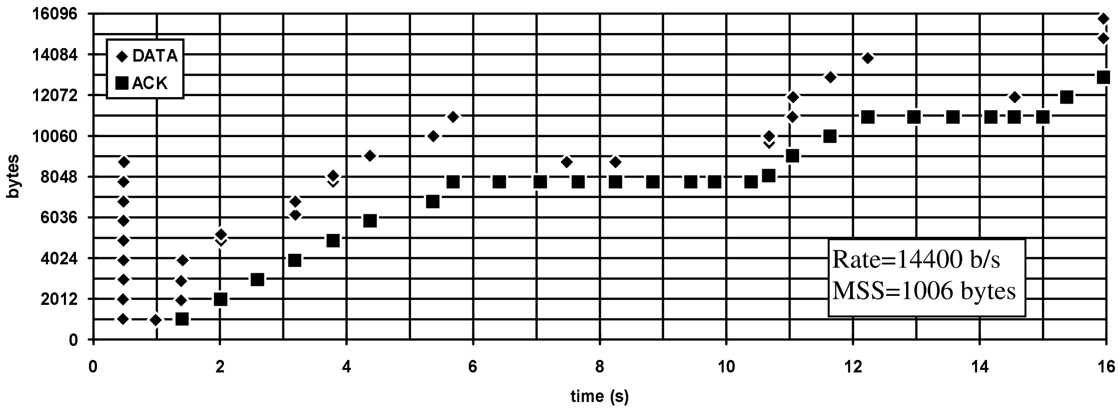

Figure 2.5. - The magnified ULTRIX papundek.fesb.hr trace

The burst of ten segments (first dot is for segments 967 and 1025) can be seen at t=0.5s. ULTRIX TCP does not implement slow start at the beginning of data transfer. RTO expires at t=1s and first segment is retransmitted shrinking transmission window to one segment. Slow start is now activated, ACK 967 initiating segments 1933 and 2899, and ACK 1025 segments 3865 and 3923. Expected window growth takes place with some silly segment packetization. TCP acts as if a whole burst of segments have been lost (go-back-n), and everything seems well until t=6s. Now ACKs for retransmitted packets arrive, indicating that segment 8753 has been lost. ULTRIX has succeeded to overflow the gateway buffer. Lost segment is retransmitted by RTO at t=7.5s and again by fast retransmit at t=8.2s. RTO and fast retransmits took place concurrently. Slow start is initiated again at t=10.6s and connection occasionally stabilizes at full speed. Other traces show that sometimes first segment is not retransmitted, or gateway buffer is not overflowed (see Figure 2.4. left).

2.3. TCP traces of LINUX zel.fesb.hr

The LINUX acceptably good trace is for R=14400 b/s and MSS=512 bytes. Somewhat worse trace is for R=14400 b/s and MSS=1460 bytes, Figure 2.6.

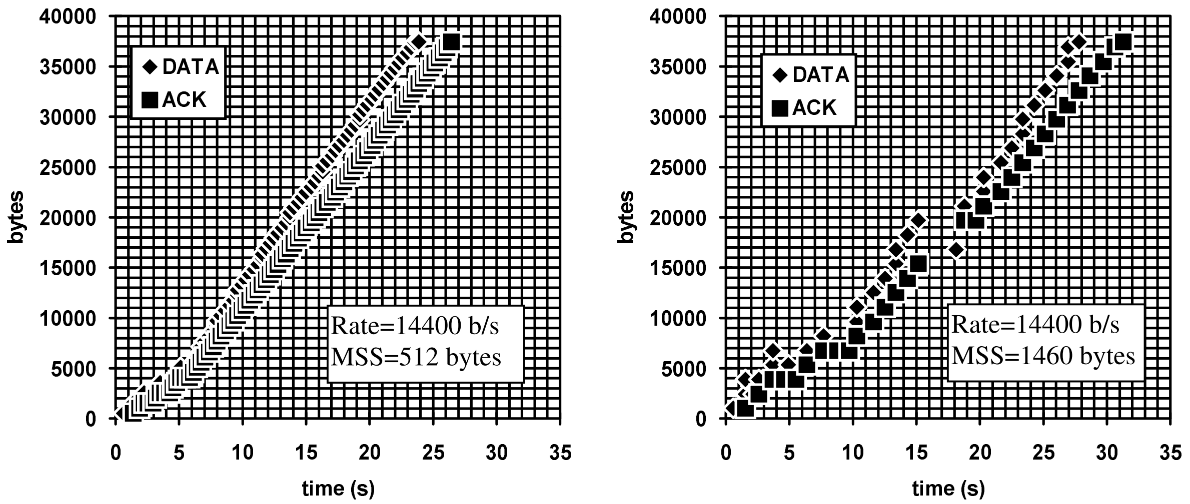

Figure 2.6. - The LINUX zel.fesb.hr traces

With MSS=512 bytes LINUX stabilizes at t=5s with few retransmissions. With MSS=1460, transfer time is about 5s longer. RTO stabilizes about t=10s, with one retransmission in the middle of the transfer. The TCP behaviour is analyzed on the magnified chart of R=14400 b/s and MSS=1460 bytes, Figure 2.7.

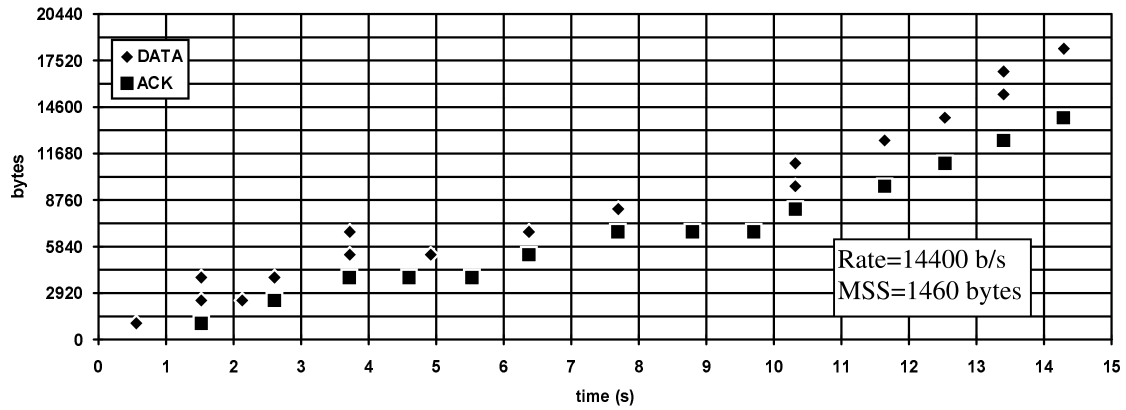

Figure 2.7. - The magnified LINUX zel.fesb.hr trace

LINUX TCP seems to follow all standard algorithms. RTO for segment 1025 does not expire, and two more segments are launched at t=1.5s. However, RTT estimation is not successful and segment 2461 retransmission takes place at t=2.1s. Slow start algorithm starts from scratch, retransmitting segment 3897 at t=2.6s. The retransmission of segment 5333 at t=6.4s shows the Karn's algorithm in effect: the RTO backoff from segment 2461 has been kept. Slow start is invoked again, and no more retransmissions have occurred. Fast retransmits have never been observed.

3. TCP PARAMETERS TUNING

Analysis of traces has shown that deficiencies in TCP implementations as well as in installations exist. In SunOS 2.4. TCP, Karn's algorithm should be implemented. Initial RTO needs to be modified. ULTRIX is characterized by lack of slow start algorithm (in the beginning of connection), and it should be implemented. The LINUX traces of full standard behaviour show that some algorithm improvements are required.

On some traces we have observed the gateway buffer overflow, and on others the slow start expansion to full receiver's window (set to 10*MSS) had caused RTO retransmissions. Receiver's window should be corrected. In later measurements, it has been scaled down to 4*MSS.

At the time of writing, only NDD utility of SunOS was available to the authors. NDD can be used to change some TCP parameters, including initial and minimal RTO values for TCP connections. We decided to set initial RTO value to 3s, and minimal to 1s, which is in accordance with [15]. Recommendation [15] implies that poor serial line performance has been observed earlier, but no real effort to achieve appropriate solution has been made (e.g. implementation of Karn's algorithm). Traces captured after the parameters had been changed are shown in Figure 3.1.

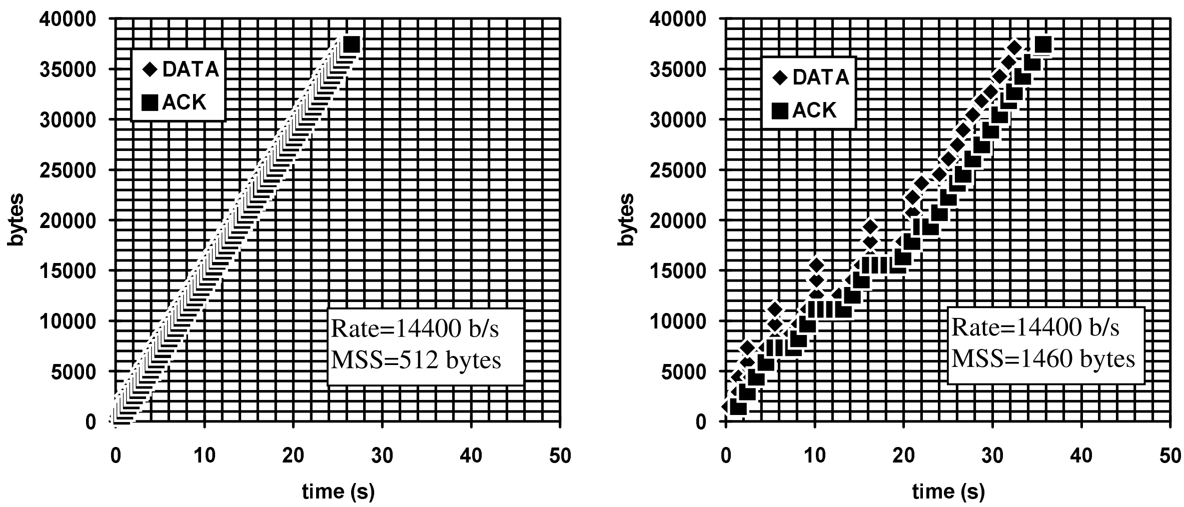

Figure 3.1. - The Solaris adria.fesb.hr traces after parameters change

With MSS=512 data transfer proceeds with theoretical maximum speed. Setting MSS=1460, RTO stabilizes at t=22s and after 19 kbytes of data were transferred. In both cases one can observe significant improvement compared with the results presented on Figure 2.2. TCP behaviour is analyzed on the magnified chart of R=9600 b/s and MSS=1460 bytes, Figure 3.2.

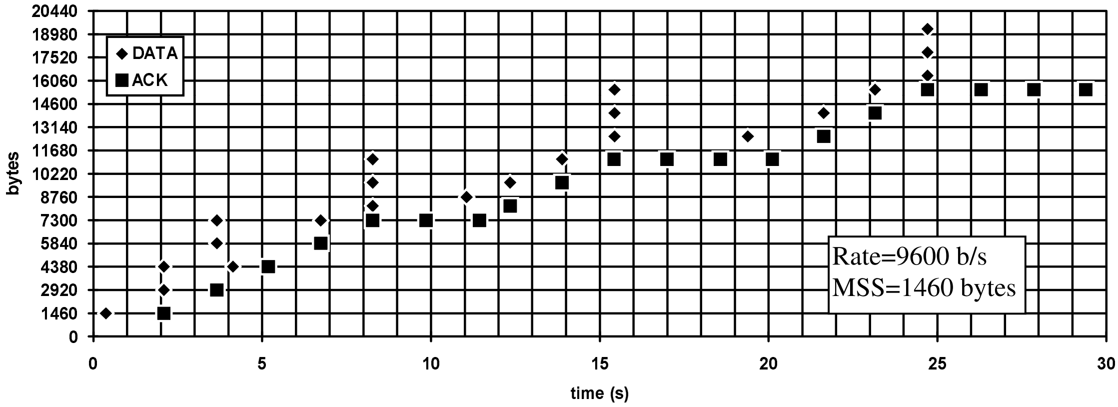

Figure 3.2. - The magnified Solaris adria.fesb.hr trace after parameters change

First and second segments are not retransmitted. However, segment 4380 has to wait for its predecessor to be transferred, so RTO expires at t=3.7s. Situation is better here, compared with Figure 2.3, but Karn's algorithm can not be substituted by pure RTO and window settings.

On ULTRIX and LINUX, shrinking the receiver's window to 4*MSS has resulted with improvements similar to those in Figure 3.1.

4. TCP ALGORITHM ANALYSIS

The general conclusion regarding the analysis in Chapter 2. is that TCP implementations suffer from weak behaviour when operating over serial lines. Despite some obvious deficiencies (lack of Karn's algorithm at SunOS and slow start at ULTRIX), the same pattern of malfunction is repeatedly observed on most of the traces. Since only LINUX TCP follows the standardization in full extent, let us concentrate on the trace in Figure 2.7.

First segment (1025) is not retransmitted because the initial RTO was set properly. However, first RTT measurement was filtered with initial RTT of zero [1], and segment 2461 is retransmitted unnecessarily by RTO 0.6s after its first instance. This should not be a problem on a complex network, but on the customer's serial line a positive feedback can be observed. Since original segment 2461 was not lost, its retransmission effectively doubles RTT. Segment 3897 is retransmitted, again unnecessarily, by slow start algorithm. That means segment 5333 has triple RTT, one by itself and two by segments queued in front of it. Retransmission of this segment at t=5s is unavoidable, which in consequence triggers slow start and one more unnecessary retransmission (of segment 6769). Finally, exponential RTO backoff wins the race and segment 8250 is not retransmitted. Still, it took almost 10s to transfer first 8k bytes of data, lowering the throughput to about 50%. Rest of data are transferred under congestion avoidance algorithm, possibly at full speed.

The mechanism of “RTT buildup” can be defined. Sending segments over slow serial line at rate greater than line bandwidth, TCP transmitter fills the buffer at the congested gateway, increasing RTT for every segment by the time needed to transfer data queued in front of it. Due to the slow start algorithm, RTT is increased by one transfer period (which is in this case equal to RTT_0) in each step. RTT rises too fast for RTT estimator to follow, and retransmissions are unavoidable.

This situation is made worse by the fact that RTO retransmissions are unnecessary, because original segments are not lost. The line is further congested with original and retransmitted segments passing through. RTT in this case increases linearly with number of retransmitted segments. The bandwidth used by unnecessary retransmissions is lost.

RTT buildup indicates that current standardization is not adequate. The behaviour of TCP is affected greatly by initial values of RTT and RTO, as well as of TCP receiver’s window. Since these are fixed at installation time, we can expect situations described above to arise repeatedly. Some more adaptive algorithm is needed for TCP connection startup. That should be considered in future research.

5. CONCLUSIONS

The series of TCP traces is analyzed in the context of present protocol standardization. It is evident that some TCP implementations do not follow standardization in full extent. All traces show similar TCP behaviour over slow serial lines, resulting in numerous unnecessary segment retransmissions.

Trace analysis was used to tune TCP working parameters where possible. The minimal and initial RTO on Solaris were increased to 1s and 3s respectively, and receiver’s window was shrunk to 4*MSS. The throughput has increased in some extent, but the fundamental TCP inadequacy for slow lines could not be compensated by pure parameter tuning.

The mechanism of “RTT buildup” is defined as RTT increase when TCP transmitter fills the gateway buffer. This effect takes place during the slow start phase of TCP connection, where exponential growth of segments sent is expected. RTT estimator can not respond to RTT buildup promptly, and some retransmissions are unavoidable. Since original segments are not lost, RTT is further increased, and instability period is prolonged. Unnecessary retransmissions act as positive feedback to the stability of connection. Exponential RTO backoff, with Karn’s algorithm, finally leads to proper RTT estimation and connection stabilization.

Described effect indicates that behaviour of TCP as currently standardized is not adequate for serial line transmission. It is affected greatly by initial values of working parameters, which are fixed at installation time. The impact of described malfunction can be significant on complex networks too, because of the "bottleneck model" of network behaviour. The more adaptive algorithm for TCP connection startup should be considered in future research.

Literature:

[1] Braden, R. "Requirements for Internet Hosts - Communication Layers", RFC 1122, IETF, October 1989, ftp://ds.internic.net/rfc/rfc1122.txt

[2] Brakmo, L.S., S.O'M., L.P. "TCP Vegas: New Techniques for Congestion Detection and Avoidance", Proceedings of ACM SIGCOMM'94, pp 24-35, ACM May 1994, ftp://ftp.cs.arizona.edu/xkernel/Papers/vegas.ps

[3] Clark, D. "Window and Acknowledgment Strategy in TCP", RFC 813, IETF, July 1982, ftp://ds.internic.net/rfc/rfc813.txt

[4] Jacobson, V. "Congestion Avoidance and Control", Proceedings of ACM SIGCOMM'88, pp. 314-429, ACM August 1988.

[5] Jacobson, V. "Dynamic Congestion Avoidance / Control (long message)", Message to tcp-ip mailing list, February 1988, http://www-nrg.ee.lbl.gov/nrg-email.html

[6] Jacobson, V., C.L., S.McC. "TCPDUMP manual page", Linux 1.2.3. manual page, 1993.

[7] Jacobson, V., R.B., L.Z. "TCP Extensions for High-Speed Paths", RFC 1185, IETF, October 1990, ftp://ds.internic.net/rfc/rfc1185.txt

[8] Jacobson, V., R.B., L.Z. "TCP Extensions for High Performance", RFC 1323, IETF, May 1992, ftp://ds.internic.net/rfc/rfc1323.txt

[9] Karn P., C.P. "Round Trip Time Estimation", Proceedings of ACM SIGCOMM'87, pp. 2-7, ACM August 1987.

[10] Nagle, J. "Congestion Control in IP/TCP Internetworks", RFC 896, IETF, January 1984, ftp://ds.internic.net/rfc/rfc896.txt

[11] Ozegovic, J., L.P. "TCP Serial Line Efficiency Measurements on CARNet FESB Node", accepted for SOFTCOM'96, Split 1996.

[12] Postel J. "Transmission Control Protocol", RFC 793, IETF, September 1981, ftp://ds.internic.net/rfc/rfc793.txt

[13] Postel J. "The TCP Maximum Segment Size and Related Topics", RFC 879, IETF, November 1983, ftp://ds.internic.net/rfc/rfc879.txt

[14] Romkey, J. "A Nonstandard for Transmission of IP Datagrams over Serial Lines: SLIP", RFC 1055, IETF, June 1988, ftp://ds.internic.net/rfc/rfc1055.txt.

[15] xxx "TCP/IP apps have slow response slip, ppp using pcnfs", http://access1.sun.com/cgi-bin/info2.html?faqs/120202.faq

[16] Zheng, W., J.C. "A New Congestion Control Scheme: Slow Start and Search (Tri-S)", ACM Computer Communication Review, vol. 21, pp 32-43, January 1991.