On the Unsuitability of TCP RTO Estimation over Bursty Error Channels

Marta García, Ramón Agüero, Luis Muñoz

Departamento de Ingeniería de Comunicaciones – ETSIIT
Universidad de Cantabria
Avda los Castros s/n, 39005 Santader (SPAIN)
`{marta, ramon, luis}@tlmat.unican.es`

Abstract. This work identifies the presence of long idle times as the main cause for the high performance degradation suffered by TCP over bursty error environments. After a comprehensive and fully experimental analysis, performed over an IEEE 802.11b real platform, it is derived that the traditional computation that TCP uses for the RTO estimation does not behave properly over channels prone to suffer from bursty errors. The authors propose a modification to that algorithm so as to avoid such an undesirable behavior

1 Introduction and Objectives

INTERNET transport protocols were originally designed to work appropriately over traditional wired links, where losses of packets were mainly caused by the overwhelming of intermediate routers. By contrast, wireless channels are likely to damage some packets due to the hostile characteristics of the radio link. The original design and implementations of TCP were not targeted to behave well with these conditions, and several methods to overcome this drawback have been studied and proposed [1], [2]. Most of them focus on the TCP congestion control procedures and sometimes they have lead to new TCP versions, as the New Reno case [3], in which only a slight change was made to the Fast Recovery algorithm used on the Reno version. This work thoroughly analyzes the impact of wireless errors over TCP performance, by means of a completely experimental approach. In particular, the presence of long idle times at the sender side, originated by the consecutive expiration of the TCP retransmission timeout (RTO) when multiple data segments are lost within a single window, has been observed. These inactivity periods bring about a sharp decrease on the TCP throughput.

2 Influence of Error Bursts Over TCP Performance

The results presented on this section have been obtained through an experimental measurement campaign over an IEEE 802.11b single-hop, comprising of two hosts configured in ad-hoc mode. To generate TCP traffic, a 10 Mbytes file was transferred using an FTP application, resulting in around 7500 data segments, as a Maximum Segment Size (MSS) of 1448 bytes has been employed. Table I shows the measured throughput at a typical office environment with metallic obstacles and people moving around the radio channel in which the signal to noise ratio (SNR) was around 10 dB, as shown in Fig. 1.

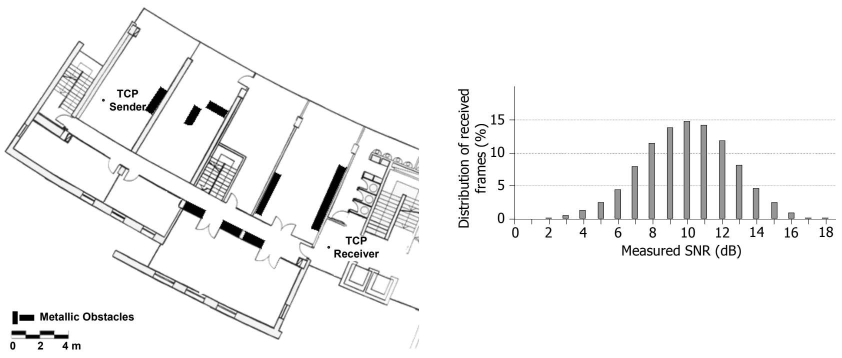

Fig. 1. Experimental environment and its SNR distribution

The IEEE 802.11 standard specifies the use of an idle repeat request (RQ) with implicit retransmission scheme at the MAC layer. In this sense, an erroneous frame might be retransmitted up to a certain number of times (three in our platform). Consequently, the IP loss, which can be defined as the number of datagrams not recovered by the IEEE 802.11 retransmission scheme, does not match the Frame Error Rate (FER). If this procedure is not enough to recover from errors, the TCP will trigger its retransmission mechanisms. All these retransmissions, triggered either by the MAC protocol or by the TCP itself may cause a performance degradation. The radio channel suffers from a extreme variability and, therefore, both erroneous frame and lost IP packet burst statistics obtained for different realizations of the experiment vary within a wide range.

Apart from the throughput, defined as the number of useful bytes received (the size of the file) over the transfer time, the following performance metrics have been collected, being reported on Table I

Retransmissions: the total number of TCP data segments retransmitted by the TCP protocol sender entity.

# Max Retx: maximum number of retransmissions for the same TCP data segment.

RTT: Round Trip Time.

Idle time: period of inactivity at the sender.

FER: the ratio between the number of medium access control (MAC) frames received with cyclic redundancy check (CRC) error and the total number of received frames.

IP Loss: percentage of datagrams that have not been recovered by the 802.11 retransmission scheme.

Average Erroneous Frame/Lost IP Packet Burst: average length of erroneous frame/lost IP packet bursts.

Maximum Erroneous Frame/Lost IP Packet Burst: maximum length of all the erroneous frame/lost IP packet bursts.

TABLE I. TCP PARAMETER REPORT OVER AN IEEE 802.11B LINK (11 MBPS) IN A BURST-ERROR ENVIRONMENT

#Test	Tput (Mbps)	FER	Avg EFB	Max EFB	IP loss	Avg LPB	Max LPB	TCP Rtx	Max Rtx	Avg RTT (ms)	StdDev RTT (ms)	MaxIdle Time (sec)	TotalIdle Time (sec)
1	2.91	0.135	1.8	29	1.20%	1.73	7	89	5	42.11	16.95	1.6	10.73
2	0.71	0.157	2.4	40	2.30%	2.27	10	178	8	41.83	35.6	57.4	102.23
3	4.49	0.058	1.1	4	0.03%	1.00	1	2	1	50.05	11.35	0.17	1.15
4	0.50	0.097	2.2	50	1.50%	2.54	12	112	6	45.86	44.74	120	152.83
5	4.59	0.004	1.4	4	0.04%	1.00	1	3	1	47.43	11.09	0.4	1.66

The achieved throughput can be roughly categorized into three different levels: poor, which is below 1 Mbps (tests #2, #4); medium, around 3 Mbps (test #1); and good, close to 5 Mbps (tests #3, #5), which the maximum throughput obtained over an error-free channel [4].

The dominant factor in the performance reduction for the tests belonging to the poor level is the presence of idle times. These are caused by the congestion control procedures implemented within TCP. It is worthwhile to perform a deeper analysis of these idle times, allowing to identify their specific causes and effects.

The measurement campaign was performed using Linux operating system (OS). The TCP implementation followed the well known Reno type, although some slight modifications were also used, as will be further explained [5]. Moreover, the Timestamp option was set during all the measurements, allowing us to track the variables used by TCP to manage its retransmission procedures. In addition, the selective acknowledgment option (SACK), which has proved to be the most effective way of dealing with multiple errors within the same window [6], was also activated during the experiments.

It is well-known that a TCP sender adapts its transmission rate to the state of the network; in this sense, it can only send a certain number of segments before receiving an acknowledgment, being this number indicated by the sender congestion window. Afterwards, it stops and if no confirmation is received within a period of time, it retransmits the first unacknowledged segment. A thorough analysis of the whole set of functions that are used to control the corresponding timers and variables is out of the scope of this text, but it is useful to give a brief introduction about how they are handled. The variable which is used to set the timeout value for a retransmission is the retransmission timeout (rto within the Linux OS), derived from both the mean value and standard deviation of the round trip time (srtt and mdev, respectively).

As the Timestamp option is activated, both srtt and mdev are updated each time new data is acknowledged. Furthermore, and following Nagle's algorithm [7], a TCP sender is not allowed to put any more new data on the channel as far as it has previously retransmitted some segments (by RTO expiration) and it still has more data to be confirmed by the receiver. On the other hand, and provided that the two aforementioned conditions are true (i.e. the TCP sender is in the loss state), the RTO is updated each time a new ACK is received, being multiplied by a backoff factor, whose value is doubled whenever a retransmission is triggered by timeout expiration. Fig. 2 shows the interchange of TCP segments between the two hosts, leading to an inactivity period on the TCP transmitter of 57 seconds (test #2 in Table I), being this the main reason for the throughput degradation within this particular measurement.

We can roughly follow the variation that the aforementioned variables have during this particular segment interchange, to see whether the 57 seconds inactivity period is due to the particularities of the TCP implementation. This is summarized on Table II, and a more detailed explanation follows. At the beginning of this particular connection chunk, the contention window managed by the TCP sender (snd_cwnd) was eight. After sending the 8th segment, the TCP sender stops and waits either for an ACK or the RTO expiration. On that moment, the RTO was at its minimum allowed value (200 ms for this Linux TCP implementation [5]) so, given that no ACK is received within 200 ms, the 1st segment is retransmitted. Furthermore, and following the slow start procedure, the value of snd_cwnd drops to one. However, as can be observed, the TCP sender does not receive confirmation that this segment arrived correctly at the receiver, so it has to be retransmitted again up to four times (each of them doubling the RTO, as specified in the backoff procedure).

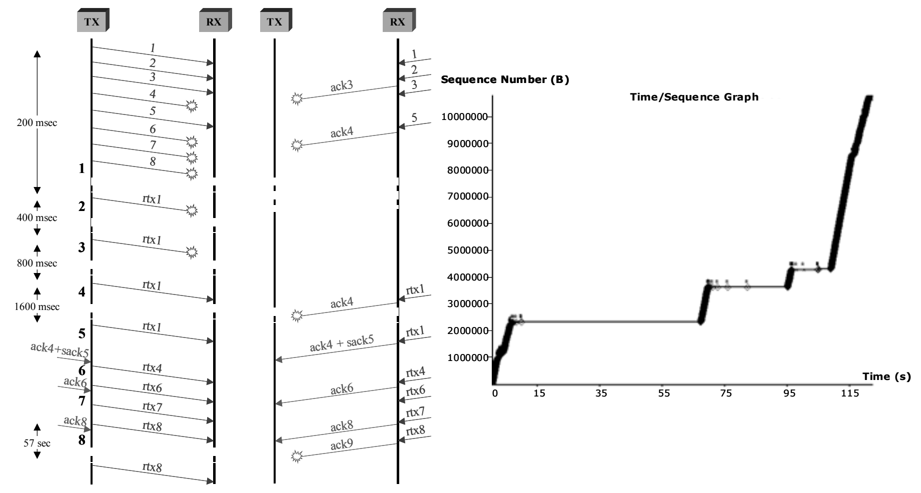

Fig. 2 Segment interchange and idle time of 57 seconds on a TCP connection

TABLE II. EVOLUTION OF THE VARIABLES IN A TCP CONNECTION LEADING TO A 57-SECOND IDLE TIME

Situation	Event	srtt (ms)	mdev (ms)	snd_cwnd	Snd_ssthresh	retransmits	backoff	packets_out
1	Start	30	10	8	?	0	0	8
2	Rtx 1 (rto)	30	10	1	4	1	1	8
3	Rtx 1 (rto)	30	10	1	4	2	2	8
4	Rtx 1 (rto)	30	10	1	4	3	3	8
5	Rtx 1 (rto)	30	10	1	4	4	4	8
6	Rx ack(4)	400	750	2	4	4	4	5

7	Rx ack(6)	350	660	3	4	4	4	3
8	Rx ack(8)	310	580	4	4	4	4	1

After the reception of the acknowledgment of that 1st segment (situation #6), the TCP sender still has more data to be confirmed by the receiver, and therefore, the calculation of the RTO is done using the previous backoff value (which was 4), although the segment that caused it to increase has already been confirmed.

The retransmissions of the 4th, 6th, 7th and 8th segments don't affect the set of variables which are being analyzed, and are triggered upon the reception of a new ACK while being retransmitting. These new acknowledgments cause the snd_cwnd to increase up to 4.

When the acknowledgment for the 7th segment is received, RTO's value is updated to 57 seconds, as observed in Fig. 2. This value is derived from the following expression, applying the corresponding values (situation #8 in Table II) for all the variables:

$$rto = (srtt + 4mdev)\left(1 + \frac{1}{4} + \frac{1}{2^{snd\_cwnd-1}}\right)2^{backoff} \quad \mathbf{(1)}$$

This expression is slightly different from the one which is specified in the standard (multiplied by a 5/4 and by a factor that depends on the contention window), so that it is more suitable for the Linux OS characteristics.

Unfortunately, the acknowledgment for this retransmission is lost (that is, four consecutive 802.11 frames arrived erroneously at the transmitter) and therefore the 8th segment is retransmitted again 57 seconds after its previous attempt. It is important to remark that the TCP sender still stays at the loss state so, despite not having filled the congestion window, it is not able to transmit any new segment. Moreover, it does not seem reasonable to maintain the backoff factor upon the reception of new acknowledgments, even though the segment that brought about this high value has already been confirmed. Hence, it is likely that modifying this approach, by cleaning the backoff variable upon the acknowledgment of the first lost segment, will help to alleviate the low throughput that has been observed.

3 Conclusion

In this work a deep analysis of the behavior of TCP over Wireless LAN has been performed, with special attention on the impact of wireless error bursts over its performance. It goes without saying that a lot of researching effort is being put on the enhancement of TCP wireless performance. However, most of the studies lack from an experimental approach and assume a TCP acknowledgment error free arrival, focusing on the improvement of the Fast Recovery algorithm. In this work we have shown that, on a real scenario, TCP acknowledgment losses can not be neglected, as they may lead to long idle times, causing a high performance degradation.

This study can be seen as a contribution towards the identification of new modifications to be done to the TCP RTO estimation procedure, adapting it to hostile wireless channels. In particular, resetting the backoff variable after the acknowledgement of the segment that caused it, might bring substantial improvements on the performance.

Although this work has been carried out over an IEEE 802.11b platform, its results can be easily extrapolated to other wireless infrastructures, such as IEEE 802.11g or Universal Mobile Telecommunications System (UMTS)

References

1. A. Chockalingam, M. Zorzi, and R. R. Rao, "Performance of TCP on Wireless Fading Links with Memory," in Proc. IEEE ICC´98, vol. 2. Atlanta, June 1998, pp. 595-600.
2. Yang, R. Wang, F. Wang, M. Y. Sanadidi, and M. Gerla, "TCPW with Bulk Repeat in Next Generation Wireless Networks," in Proc. ICC´03. Anchorage, Alaska, May 2003.
3. Floyd and T. Henderson, "The NewReno Modification to TCP´s Fast Recovery Algorithm, RFC 2582," April 1999.
4. Muñoz, M. García, J. Choque, R. Agüero, and P. Mähönen, "Optimizing Internet Flows over IEEE 802.11b Wireless Local Area Networks: A Performance Enhancing Proxy Based on Forward Error Correction," IEEE Communications Magazine, vol. 39, nº 12, pp. 60-67, December 2001.
5. Sarolahti and A. Kuznetsov, "Congestion Control in Linux TCP," in Proc Usenix 2002. Monterey, CA, USA, June 2002, pp. 49-62.
6. Fall and S. Floyd, "Simulation-based Comparisons of Tahoe, Reno, and SACK TCP," ACM Computer Communication Review, vol. 25 nº 3, pp. 5-21, July 1996.
7. Nagle, "Congestion Control in IP / TCP Internetworks, RFC 896," January 1998.