Avoiding Spurious TCP Timeouts in Wireless Networks by Delay Injection

Thierry E. Klein Kin K. Leung
Wireless Research Laboratory
Bell Laboratories - Lucent Technologies
600 - 700 Mountain Avenue
Murray Hill, NJ 07974, USA
e-mail: tek, kin@lucent.com

Richard Parkinson Louis G. Samuel
Wireless Research Laboratory
Bell Laboratories - Lucent Technologies
The Quadrant Stonehill Green
Swindon, SN5 7DJ, Un ited Kingdom
e-mail: rp15, lsamuel@lucent.com

Abstract — **The transmission control protocol (TCP) has been designed to provide reliable transport of packets by adjusting the transmission rate to the network congestion level. While TCP can adapt to small fluctuations in the delay between the sender and the receiver, adverse affects (most importantly spurious timeouts) have been observed under large delay variability. In this paper, we exhibit the presence of such delay spikes in wireless networks and discuss their possible origins. We then investigate a new methodology for avoiding spurious TCP timeouts by appropriately injecting additional random delay along the communication path. Different algorithms for the delay injection are presented and we assess their relative performances and merits through simulations. In particular we show by numerical examples that the delay injection methodology can significantly decrease the number of timeouts and increase the achieved TCP throughput by about 8% in the network scenario considered in this paper. One of the attractive features of the new methodology is that it does not require any changes to the TCP protocol and can be applied independently of the TCP version used.**

Keywords: **Wireless networks, TCP, timeout, throughput, end-to-end performance.**

I. Introduction

The transmission control protocol (TCP) remains the most widely used transport control protocol in the Internet today. Although TCP was initially designed and optimized for wireline networks, with the growing popularity of wireless data applications, it is increasingly being used over wireless networks as well. The main objective of TCP is to efficiently use the available bandwidth in the network and to prevent overloading the network (and the resulting packet losses) by appropriately throttling the senders' transmission rates. Network congestion is deemed to be the underlying reason for packet losses. This is in sharp contrast to wireless networks in which packet losses may occur for various other reasons related to the time-varying nature of the wireless channel and the mobility of the end user. As a consequence, TCP may interpret packet losses due to transmission errors, high latency and delay variability as indications of network congestion and react inappropriately by reducing the sender's transmission window and initiating its slow start phase and the congestion control mechanism. Most wireless networks aim at avoiding packet losses at the TCP layer by implementing robust link layer protocols (using error control coding and robust link adaptation) as well as soft handoff and seamless mobility management algorithms. However all of these solutions increase the packet transmission delay and its variability and may have adverse effects on the TCP behavior.

In this paper, we concentrate on the issue of *delay spikes* and their influence on TCP timeouts and throughput. Delay spikes are defined as a sudden and significant change in the round-trip time between a TCP sender and its receiver. High delay variability has been observed in fixed wired networks and can be caused for example by route flipping [1]. In wireless networks on the other hand, the delay variability can be attributed to several factors, most notably the time-varying quality of the wireless link (both the inherent variability and the one induced by the terminal mobility). A sudden change in the link quality leads to a burst of transmission errors and link level retransmissions. The retransmissions in return cause an increase in the packet transmission delay. Other reasons for delay spikes may include handoff delay when users are transferred between receiving base stations and transmission interruptions due to priority scheduling and preemptive service. The latter reason is becoming increasingly important as future wireless networks become more and more multimedia networks, requiring efficient scheduling algorithms to maintain quality of service guarantees.

Delay spikes have been observed and measured independently by several researchers [7], [9] and [12]. The effects of large delays and delay variability on the behavior of TCP have been investigated in [3] and [4]. In particular it is shown that sudden increases in the delay may lead to spurious TCP timeouts and trigger two undesirable responses [10]. First TCP interprets the timeout as being caused by packet losses and (unnecessarily) retransmits the packets that are presumed lost. In addition the congestion avoidance mechanism is triggered and falsely reduces the TCP window size leading to low throughput. Several solutions have been proposed to alleviate the effect of delay spikes on TCP performance. Most notably, TCP Eifel [10] has been proposed to detect spurious timeouts as well as spurious retransmits by implementing time stamping at both the sender and the receiver. However, TCP Eifel has not yet been widely deployed and also requires an additional 12-byte overhead in the TCP header. Other potential solutions are described in [5] and [6]. It is important to point out that all of the above solutions require modifications to the TCP protocol which may not be widely available. These solutions do not attempt to avoid spurious timeouts but rather change the reaction of TCP when such timeouts occur. Moreover, some of them interfere with the nature of the protocol paradigm (e.g. a split-TCP solution using a proxy mechanism).

In this paper, we take a completely different approach, and investigate a solution that does not require any change to the TCP protocol and does not break the protocol paradigm, but instead attempts to avoid triggering the TCP timeout mechanism unnecessarily. The fundamental idea is to artificially inject additional delay in the round-trip path in order to increase the timeout threshold. In other words, we provide a methodology to increase the variance of the round-trip time without significantly increasing the average round-trip time. It is worthy pointing out that the delay injection method is rather counter intuitive. Indeed the spurious TCP timeouts occur due to the high variability of the packet round trip times (RTT), but yet our new solution is to further increase the variability so that unnecessary timeouts are avoided. The original concept of TCP timeout avoidance by delay injection has been proposed earlier [2], but was not investigated in any systematic way.

Our main contribution is to quantify the potential benefits of this solution, to study different methods for performing the delay injection and to determine the respective optimal parameters for each method in a systematic way. We demonstrate that, by appropriate choice of the parameters of the injected delay, the number of timeout occurrences can be significantly reduced, if not eliminated. At the same time, the achieved TCP throughput can be increased in the cases under consideration, thereby providing improved end-user perceived performance. Besides describing this novel methodology for avoiding spurious timeouts in TCP and emphasizing the fact that this methodology does not require any TCP modifications, this paper provides a proof of concept of the so-called delay jitter algorithm and illustrates the potential throughput performance gains that may be achieved.

The remainder of the paper is organized as follows. In Section II, we provide additional details on the issue of the delay spikes and their impact on TCP. In Section III, we describe the methodology of artificial delay injection and propose several algorithms to achieve the stated goals of reducing the number of spurious TCP timeouts and increasing the TCP throughput. Performance results are presented in Section IV and the relative merits of the different algorithms are discussed. Conclusions follow in Section V.

II. Delay Spikes and Spurious Timeouts in TCP

In this section, we provide more details on the delay spikes and their impact on TCP. It is assumed that the reader is familiar with the main features of TCP and we only summarize the essence of the specific features needed for the understanding of this paper. In order to detect packet or acknowledgment losses, TCP implements a timer which can be viewed as an upper bound on the round-trip time between the sender and the receiver. If an acknowledgement is not received before the timer expires, the corresponding packet is deemed to be lost and the congestion control mechanism is triggered and the packet is retransmitted. The calculation of the timeout threshold is therefore of crucial importance [8]. Denote by $RTT[k]$ the k-th measurement value of the round-trip time.[1] The round-trip time is calculated by a timer which is started when the packet is transmitted by the TCP sender and stopped when an acknowledgement for the same packet is received. $S[k]$ denotes the *smoothed average round-trip time* and is calculated as [11]:

$$S[k] = (1-g)\, S[k-1] + g\, RTT[k] \tag{1}$$

where g is the smoothing factor with typical value $g = \frac{1}{8}$. In other words, the smoothed value of RTT is updated whenever a new RTT measurement is available. In addition the variations of the RTT are tracked by calculating the mean deviation (as an approximation of the standard deviation) [11]:

$$V[k] = (1-h)V[k-1] + h\ RTT[k] - S[k] \tag{2}$$

where the gain h is typically taken as $h = \frac{1}{4}$. Finally the timeout threshold value RTO is set to [11]:

$$RTO[k] = S[k] + m\, V[k] \tag{3}$$

with the standard choice of $m = 4$. That is, after sending a packet, the TCP sender sets a timer with the RTO value in equation (3) as the timeout threshold value. A TCP timeout is declared if an acknowledgement for the corresponding packet is not received before the timer expires (*i.e.* if the RTT associated with the packet exceeds the RTO threshold). It is expected that this usually happens when a packet or the corresponding acknowledgement is lost. If the packet is indeed lost, then the timeout mechanism is required in order to avoid deadlock. On the other hand, if the packet is not lost, but merely delayed, the timeout mechanism is falsely triggered. This timeout is called spurious and would have been avoided if the RTO value had been larger. It is easily seen that sudden and large changes in the round-trip time, *i.e., delay spikes*, can cause spurious timeouts and unnecessarily shut down the congestion window leading to unnecessary loss in TCP throughput.

In order to avoid spurious timeouts, several changes to the TCP protocol have been proposed, such as implementing the time-stamping option or changing the granularity of the TCP timer. Other solutions would aim at making the RTO calculation more robust to delay spikes and would include changing the smoothing parameters g and h and the weighting factor of the variance in the RTO calculation (*i.e.*, the parameter m). However since all of these solutions require a change in the TCP

[1]In most implementations, only one packet at any given time is tracked for the calculation of the timeout threshold. Thus the threshold is only updated once per window of transmitted packets. If the time-stamping option however is enabled, it is possible to track the round trip time of all packets.

protocol (and the related standard and the de facto standards of the control parameters) and may not be widely and readily available, it may be preferable to seek solutions that are transparent to TCP and do not depend on the TCP version or its implementation. This is the objective of the remainder of this paper.

III. Delay Jitter Algorithm

In this section, our proposed methodology for reducing the number of TCP timeouts is described in detail. We have argued against changing the TCP protocol and therefore the update equations for the calculation of RTO. The only parameters that influence the value of RTO (and thus whether there is a timeout or not) are the smoothed average and mean deviation of the RTT. Clearly we do not wish to modify the average round-trip time as the TCP throughput (for long-lived flows in steady-state) is essentially inversely proportional to the average RTT. Hence the only remaining possibility is to influence the calculation of the mean deviation (or the variance of the RTT). The fundamental idea behind this methodology (referred to as *delay jitter algorithm*) is to inject additional random delay (by holding back a packet or acknowledgment somewhere along the round-trip path) so as to increase the variance of the RTT, without significantly increasing its average value.[2]

Our contribution is to develop systematic ways and investigate different algorithms to inject additional delay in the round-trip path with the objective of reducing the number of spurious TCP timeouts caused by delay spikes and increasing the achieved TCP throughput. There is an interesting tradeoff involved in this methodology. On the one hand, reducing the number of spurious timeouts can potentially increase the throughput. On the other hand, injecting additional delay in the round trip path increases the average RTT, which in return leads to a decrease in throughput. However we demonstrate by numerical examples that, by appropriate choice of the delay jitter parameters, the net effect is an overall increase in TCP throughput along with a significant reduction in the number of timeout occurrences.

Let $D[k]$ be artificial delay injected for measurement k. Then the new value of RTT (used for the calculation of S and V and therefore for RTO) is given by $RTT_{new}[k] = RTT[k] + D[k]$. The new timeout threshold value $RTO_{new}[k]$ is calculated according to the same formulas in equations (1) to (3) as before with the corresponding value of $RTT_{new}[k]$.

We now propose several algorithms and methods for choosing the artificial delay. A constant delay $D[k]$ for every measurement (*i.e.* tracked packet) would increase the average RTT but would not affect the variance in the RTO calculation. Since not all transmitted packets are subjected to this delay injection, it would still provide some level of robustness against delay spikes. However, for added generality, random delay injection is considered where the randomness may come from either the time of the delay injection or from the actual value of the additional delay. The first algorithm called the *Fixed Time - Fixed Delay (FTFD)* method, injects a fixed delay according to a fixed schedule. In other words, we have that $D[k] = D_0$ whenever $k = i\,N$ and $D[k] = 0$ otherwise. D_0 is value of the added delay and N is the period of the schedule (the so-called *jitter period*), meaning that an additional delay of D_0 is added every N tracked packets.

A second method, called the *Random Time - Fixed Delay (RTFD)* method, injects the additional delay to every packet with a certain probability p. This means that for every measurement k, $D[k] = D_0$ with probability p and $D[k] = 0$ with probability $1 - p$. Note that by choosing $p = \frac{1}{N}$ the average additional delay (averaged over all tracked packets) is the same for both FTFD and RTFD methods.

The third method, called the *Random Time - Random Delay (RTRD)* method, introduces even more randomness in the delay injection by making the time of the injection as well as the amount of additional delay random variables. Let $f_D(d)$ be a probability density function on the additional delay with mean D_0. Then with probability p, $D[k] = d$ where d is chosen according to the above probability density function (taken to be exponential in our simulations). Conversely, with probability $1 - p$, $D[k] = 0$. All of the above proposals are parameterized by different tunable parameters that have to chosen in order to obtain the best performance.

IV. Performance Evaluation

In this section, we evaluate the performances of the methods described in Section III. To start however, we present some measurements to exhibit the presence of delay spikes in a live wireless network and motivate the use of the delay jitter algorithm. Ping tests with a pre-specified payload size were conducted on a commercial GPRS network and the round-trip time between the mobile terminal and a target router recorded. The mobile user is assumed to be stationary and all of the experiments were conducted during light network loading conditions. The target router in the network was identified using the *tracert* utility and chosen to be the first router recorded in the tracert list. This minimizes the contribution of the backhaul network to the round-trip time recorded for the pings. Consequently the major part of the recorded RTT is due to the limited data rate on the air interface and the dynamic assignment of radio resources to the data flow. In our experiments, the ping utility issues 10,000 messages and waits for a maximum of 10 seconds for the associated reply message. When the reply is received or the maximum wait time expires, the next ping request message is sent. When a reply is received, the ping utility displays a statistics report which includes the RTT for the corresponding ping message.[3]

[2]Note that in practice it only makes sense to inject delay only to those packets that are being tracked for the RTO calculation.

[3]The collected measurements in the GPRS network show that the average RTT is on the order of 500 msec. Future wireless data networks are expected to have round trip times that are significantly smaller, on the order of 100 msec. We therefore scale all the measurements by a factor of 5 and present the corresponding results in this paper. However, we note that the delay jitter methodology has also been applied to

In Figure 1, we show the RTT measurements for 0-byte ping packets, as well as the associated RTO estimate, calculated from equation (3). A timeout is declared if the RTT measurement exceeds the RTO threshold for that packet. We only show a window of 100 consecutive samples and note that there are two timeout occurrences: one for packet 25 and the other one for packet 40. The main objective is to demonstrate the presence of delay spikes, even in a scenario when the mobile is stationary (and therefore the channel variations are limited and there are no mobility-induced handoffs) and the network is lightly loaded. The number of occurrences of such delay spikes is only expected to increase in scenarios when the user is mobile and the network load is increased. Furthermore, since no packets were lost during the experiments, these timeouts are spurious and attributed to the large RTT spikes observed. Since the ping packets are only sent upon reception of the previous reply message, the RTT measurements turn out to be essentially uncorrelated and the timeouts are observed as isolated incidents. If ping packets are sent successively without waiting for the corresponding replies, the RTT measurements may be more correlated and delay spikes are expected to be observed in bursts. The impact of a burst of delay spikes may depend on the TCP implementation and in particular on the retransmission mechanism upon observing a timeout (e.g. go-back-n or selective retransmission). It is therefore expected that the performance gains of the delay jitter algorithm may also depend on the particular retransmission mechanism, although we emphasize that the algorithm itself does not change and does not need to be aware of the retransmission strategy.

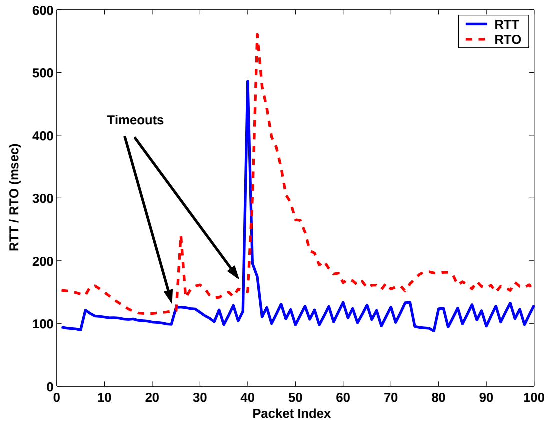

Figure 1: RTT and RTO measurements without delay jitter

We now turn our attention to the three different methods for delay injection and study their ability to reduce the number of timeouts. In the *FTFD* method, a delay of D_0 is injected every N packets where N is the period. For the *RTFD* method, the delay D_0 is injected to every packet with probability $p = \frac{1}{N}$ so that the average jitter period and the value of the delay injection remain the same. Finally, for the *RTRD* method, delay is again injected to every packet with probability $p = \frac{1}{N}$. However, when delay is injected, the value is chosen from an exponential distribution with mean D_0. First we provide some evidence that this methodology can indeed reduce the number of timeouts by applying the *FTFD* method to the vector of collected RTT measurements. The new RTO threshold is again computed according to equation (3). In Figure 2, we show the effect on the RTT measurements and RTO estimates of the *FTFD* method with $D_0 = 100$ msec and $N = 3$ (this choice for the set of parameters will be explained shortly). Note that one of the timeouts in Figure 1 has been avoided in Figure 2. However the second timeout could not be avoided as the experienced delay spike is just too large. In general, the "margin" between RTO and RTT is substantially increased, when compared to Figure 1, thus increasing the robustness to delay variations.

the original set of RTT measurements. The obtained results show that, while the number of timeouts is significantly reduced, the achieved TCP throughput is not greatly changed.

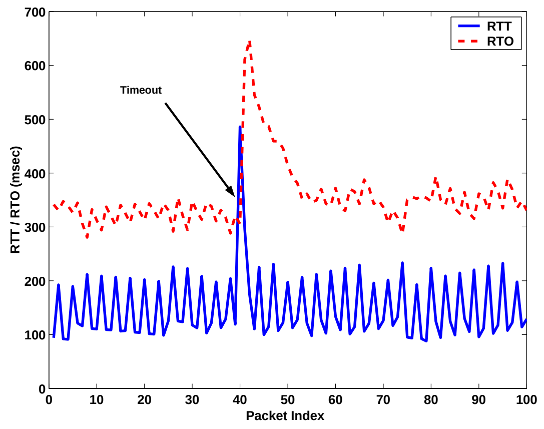

Figure 2: RTT and RTO measurements with delay jitter

Next we examine the impact of the delay injection on the distribution of the RTT measurements. The stated goal of the delay jitter method is to increase the variance of the RTT, but the resulting change on the entire distribution cannot be ignored. Figure 3 shows the RTT distribution without delay jitter and for the three different methods when the parameters are $D_0 = 100$ msec and $N = 3$. First of all, we observe that the distribution of the RTT (without delay jitter) is approximately uniform and does not exhibit the expected normal distribution. This shape is essentially preserved under a fixed delay injection, except that the distribution becomes bimodal. This could be expected since, with probability $\frac{1}{3}$ each packet is subject to an additional delay of 100 msec, which essentially corresponds to the gap between the two uniform pieces of the distribution. As could be expected, the distribution under *FTFD* and *RTFD* is essentially the same as the injection period for *RTFD* is chosen according to a geometric distribution with the same mean as the deterministic injection period for *FTFD*. However, when the injected delay is exponentially distributed, the shape of the RTT distribution is radically altered and exhibits an exponential tail. Therefore we may conclude that the delay jitter method can also be used for RTT distribution shaping.

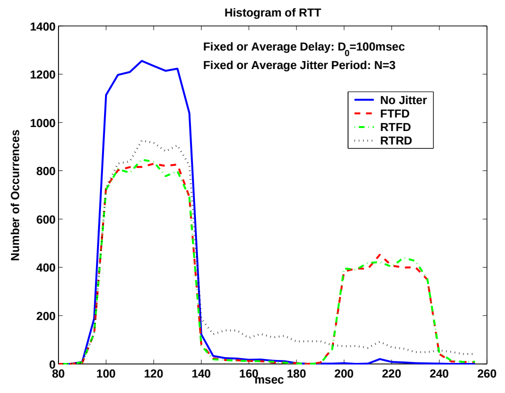

Figure 3: Histogram of RTT measurements with and without delay jitter

After justifying the use of the delay jitter algorithm by simple comparison between the RTT measurements and the RTO estimates, we now set out to assess its impact on the TCP performance. We have build a complete TCP simulator capturing all the feedback and congestion control and timeout mechanisms [11]. In particular, we implement selective repeat as retransmission policy, but disable the time stamping option. We consider four different values for the amount of delay injected $D_0 = 40, 60, 80$ and 100 msec and for each value the jitter period is varied. In Figures 4, 5, 6 and 7, we show the number of TCP timeouts under the different injection methods that were recorded by our TCP simulator during an FTP-like transfer of 512 segments of 500 bytes each. The results shown are the average values from 10 independent experiments. In all four plots, the horizontal line denotes the number of timeouts observed without the delay jitter algorithm.

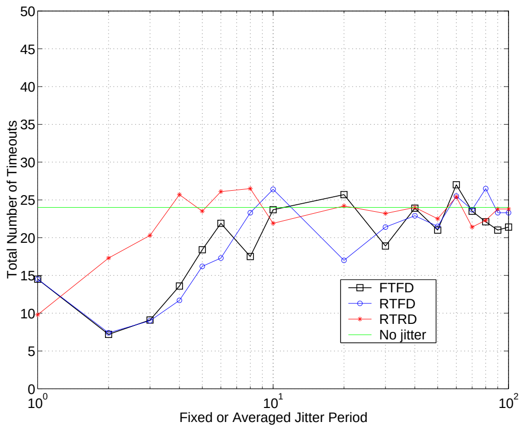

Figure 4: Number of timeouts when $D_0 = 40$ msec.

Several important conclusions may be drawn. First and foremost, it is observed that the proposed methodology can indeed reduce the number of TCP timeouts, provided that the

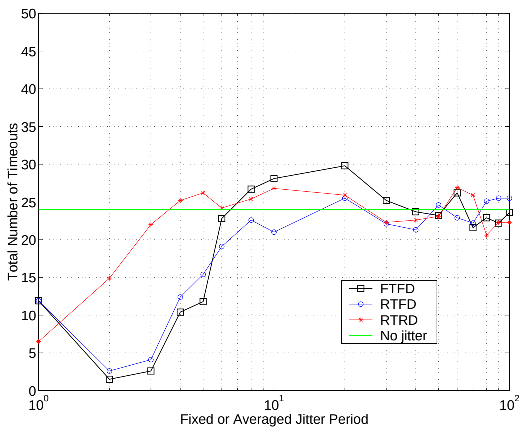

Figure 5: Number of timeouts when $D_0 = 60$ msec.

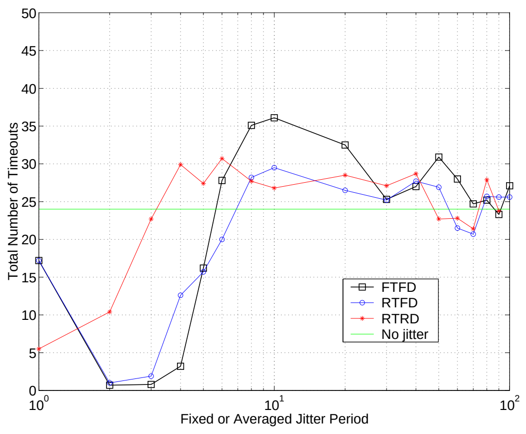

Figure 6: Number of timeouts when $D_0 = 80$ msec.

parameters are carefully chosen. This observation is valid for all three methods of delay injection. As an additional sanity check, consider the extreme case when the jitter period is very large. In this case, delay is injected so infrequently that it essentially does not affect the RTT and the corresponding RTO calculation. Therefore, we expect that the number of timeouts is essentially equal to the number of timeouts without the jitter algorithm. This trend is observed in all four plots and for all three methods and is confirmed if the jitter period is further increased. In addition, the reader might expect that, when $N = 1$, the delay is injected to every packet and consequently the RTT for every packet is increased by the same amount (at least for *FTFD* and *RTFD* methods) and therefore the number of timeouts is not changed. This is not the case though, since the delay jitter algorithm is only applied to those packets that are being tracked for the RTO calculation; in our implementation only one packet per window of transmitted packets is being tracked. The aforementioned intuition that for $N = 1$ the number of timeouts is not changed compared to the no-jitter scenario is correct if TCP tracks all packets for RTO calculation, as would be the case if the time-stamping option were enabled.

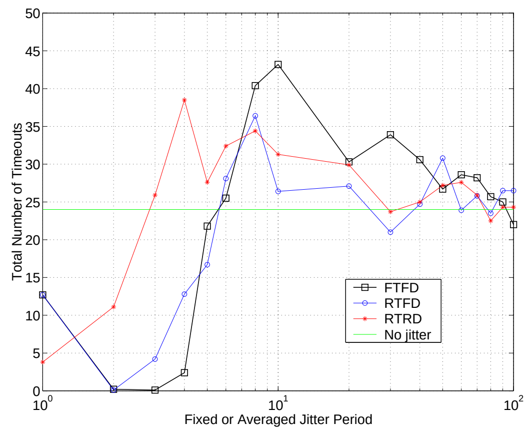

Figure 7: Number of timeouts when $D_0 = 100$ msec.

In general it is noted that the *RTRD* method performs quite poorly with respect to the other two methods. The reason for the performance degradation is that the method introduces too much randomness into the RTT measurements. In essence, randomness is desirable to increase the variance of the RTT to a certain extent, but this same randomness introduces too much uncertainty in the measurements themselves. The lesson to be learned is that the degree of randomness has to be tightly controlled for desirable results. The remaining two methods, *FTFD* and *RTFD*, effectively manage to reduce the number of TCP timeouts and thus achieve one of the stated design goals. The *FTFD* method (at least for the experiments performed in this study) gives the optimal performance in terms of reducing the number of TCP timeouts. The best performance is achieved when $D_0 = 100$ msec and $N = 3$ (reducing the average number of timeouts from 24 to 0.1).

The above results however only show one aspect of the TCP performance. In particular, users are more concerned with their achieved throughput. We now turn our attention to this second aspect and show the delay jitter algorithm leads to an overall throughput improvement. We have already observed that the delay jitter algorithm leads to a slight increase in the average RTT, which is equal to $p\, D_0\, \frac{1}{W}$, where p is the delay injection probability, D_0 is the average injected delay and W is the average window size. The latter term comes from the fact that the delay is potentially only injected to packets that are tracked for RTO calculation, i.e. to at most one packet per window of outstanding packets. This small increase in average RTT negatively affects the throughput performance, since the TCP throughput T can be approximated as the ratio of the average window size to the average RTT. In the absence of timeouts, the delay jitter algorithm would not provide any throughput gain. However the presence of delay spikes causes spurious timeouts which unnecessarily reduce the throughput. The delay jitter algorithm, besides a slight increase in the average RTT, also has the desirable effect of eliminating spurious timeouts, which would increase the throughput. In the remainder of this section, we show that the increase in RTT can be more than offset by the reduction of timeouts and the net effect of these two consequences of the delay jitter algorithm can result in an increased throughput. Since the *FTFD* method is the preferred method to reduce the number of timeouts, we exclusively concentrate on *FTFD* for our throughput analysis. In Figure 8, we plot the achieved TCP throughput as a function of the jitter period for different values of the injected delay. The throughput without the delay jitter is shown as a horizontal line and is equal to 103.58 kbps. Note that a maximum throughput of 111.96 kbps (which represents an improvement of about 8%) is achieved when $D_0 = 70$ msec and $N = 5$ (at least for the studied range of parameters). Thus we have demonstrated that the delay jitter algorithm can indeed be considered as a viable option to alleviate the effect of delay spikes and reduce the number of spurious timeouts and provide throughput gains. We point out that our TCP simulator implements the selective retransmission policy and, upon timeout, retransmits only the corresponding packet. In other implementations of TCP, such as those that use a go-back-n mechanism, all outstanding packets would be retransmitted. The inherent throughput degradation resulting from a spurious timeout is expected to be larger and consequently the improvement achieved by delay injection is expected to be more significant. This is especially true if the temporal correlation in the RTT samples is small. In that case, subsequent packets, following a packet that experienced a delay spike, do not expect to experience delay spikes themselves. On the other hand, if the temporal correlation is stronger, a go-back-n retransmission strategy may provide better results as some of the subsequent packets (which are likely to timeout as well) are pro-actively retransmitted and their timeouts in some sense anticipated.

We note that, with the set of parameters that minimizes the number of timeouts (*i.e.* $D_0 = 100$ msec and $N = 3$), the achieved throughput (not shown in the graph) is only 101.20 kbps. It is therefore interesting to observe that minimizing the number of timeouts in fact may not necessarily maximize the achieved TCP throughput. In other words, to eliminate virtually all timeouts, the average injected delay needs to be fairly large in order to make the timeout threshold more robust to delay spikes. The required delay value however results in an overall throughput degradation. A possible explanation is that TCP timeouts tend to occur in bursts (as observed by our simulations) and therefore do not all have the same effect on TCP throughput (since the congestion window remains small for subsequent timeouts after the initial timeout in the same burst).

We emphasize again that the delay jitter algorithm can be applied with any implementation and version of TCP and does not require any explicit TCP-level information and any modifications to the TCP standard. To our knowledge, this methodology has not appeared in the literature and is complementary to other advocated approaches. Finally we comment on the possible implementation of the proposed delay jitter algorithm. As pointed out earlier, there is no advantage to inject delay to packets whose

RTT is not tracked and that are therefore not considered for the RTO calculation. One could enable the TCP time-stamping option in which every packet is tracked for the RTO calculation and the proposed algorithm can be directly applied to all packets. On the other hand, without the time-stamping option, typically only one packet per window is tracked. Identifying the tracked packets to enable efficient delay injection for performance gains remains an open research problem. To inject a given delay to a packet, it can be temporarily put in a buffer for the specified amount of delay before being released to the rest of the communication path. Although this may cause packets to arrive out of order at the receiving end, TCP duplicate acknowledgements and buffering out-of-sequence packets at the receiver will ensure proper reception and protocol operation. In addition, since the optimal delay jitter parameters may depend on the underlying delay statistics and the network scenario at hand, an adaptive version of the delay jitter algorithm may prove necessary in practice for maximum performance gains. This is the objective of current investigations and goes beyond the scope of this paper, which aims at a proof of concept of this novel methodology.

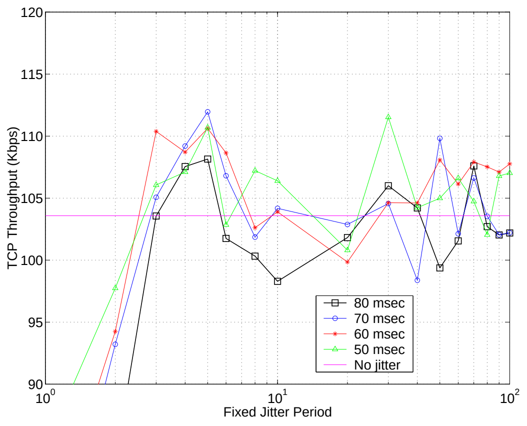

Figure 8: TCP throughput as a function of the jitter period for different values of the injected delay in the FTFD method.

V. Conclusions

In this paper, we have shown the presence of delay spikes in wireless networks, have discussed their possible origins and described their negative impact on TCP performance by causing spurious timeouts. Rather than modify the TCP protocol, we investigated an innovative methodology that significantly reduces the probability of TCP timeouts and consequently increases the TCP throughput performance. The essence of the methodology is to inject artificial delay in the round-trip path in order to increase the variance of the round trip estimate and thereby increase the timeout threshold calculation. Several algorithms for injecting delay are presented and their relative performances assessed. Numerical results demonstrate that the proposed delay jitter algorithm is a viable alternative to other methods to combat spurious timeouts and that it provides throughput gains, even in fairly simple and well-behaved network scenarios.

Acknowledgments

The authors would like to acknowledge helpful discussions with J. Cloutier and M. Sherif. The authors also thank Gee Rittenhouse and Mark Haner for their continued support of this research.

References

[1] M. Allman and V. Paxson, ”On Estimating End-To-End Network Path Properties”, in *Proceedings of ACM SIGCOMM*, September 1999.

[2] J. Cloutier *et al.*, ”Improved Wireless Data Transmission using Time Out Control”, *private communication*.

[3] S. Fu, M. Atiquzzaman and W. Ivancic, ”Effect of Delay Spike on SCTP, TCP Reno and Eifel in a Wireless Mobile Environment”, in *Proceedings of International Conference on Computer Communications and Networks*, October 2002

[4] A. Gurtov, ”Effect of Delays on TCP Performance”, in *Proceedings of IFIP Personal Wireless Communications*, August 2001.

[5] A. Gurtov, ”Making TCP Robust Against Delay Spikes”, University of Helsinki, Department of Computer Science, Technical Report C-2001-53, November 2001.

[6] A. Gurtov and R. Ludwig, ”Responding to Spurious Timeouts in TCP”, in *Proceedings of IEEE INFOCOM*, March 2003.

[7] A. Gurtov, M. Passoja, O. Aalto and M. Raitola, ”Multi-Layer Protocol Tracing in a GPRS Network”, in *Proceedings of the IEEE Vehicular Technology Conference*, September 2002.

[8] V. Jacobson, ”Congestion Avoidance and Control”, in *Proceedings of ACM SIGCOMM*, 1988.

[9] J. Korhonen, O. Aalto, A. Gurtov and H. Laamanen, ”Measured Performance of GSM HSCSD and GPRS”, in *Proceedings of the IEEE Conference on Communications*, June 2001.

[10] R. Ludwig and R. Katz, ”The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions”, in *ACM Computer Communication Review*, vol. 30, No. 1, January 2000.

[11] W. R. Stevens, *TCP/IP Illustrated, Vol. I*, Addison Wesley, 1995.

[12] M. Yavuz and F. Khafizov, ”TCP over Wireless Links with Variable Bandwidth”, in *Proceedings of the IEEE Vehicular Technology Conference*, September 2002.