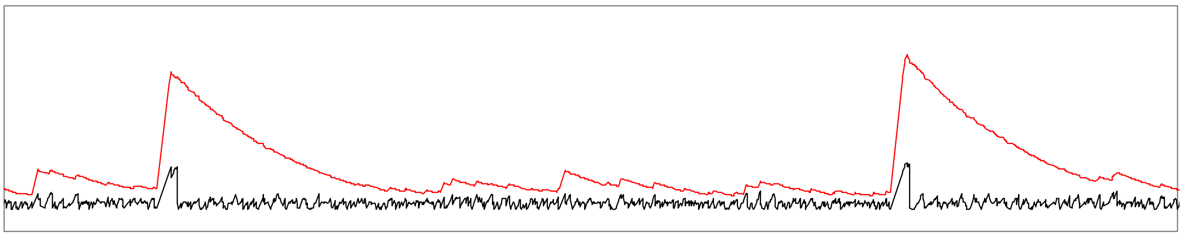
DRAFT VERSION July 2nd 2003 (paper under submission)

The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport



Hannes Ekström, Reiner Ludwig

Ericsson Research

Aachen, Germany

***Abstract*— We propose a new retransmission timer named the Peak-Hopper-Timer (PH-Timer). Based on simulations, we compare its performance to that of the current retransmission timer specified in [RFC2988]; referred to as the RFC2988-Timer. We evaluate a number of problems related to the RFC2988-Timer such as its interaction with timing every packet as specified in [RFC1323]. Our results show that in most cases the PH-Timer yields better performance both in terms of quick loss detection when a packet was lost, and in terms of minimizing the number of spurious timeouts. The PH-Timer on average detects a lost packet up to one Round-Trip Time (RTT) quicker than the RFC2988-Timer while at the same time causing less or at least not more spurious timeouts. Moreover, the loss detection times for the PH-Timer are much more closely spread around the average whereas the RFC2988-RTO often generates "RTO outliers" that lead to exceptionally long loss detection times. Furthermore, we show that a considerably better retransmission timer performance is achieved when timing every packet instead of only one packet per RTT.**

Keywords-Retransmission Timer, Performance Evaluation

I. Introduction

In the traditional approach to reliable transport, a retransmission timer is an essential component of the protocol design. With no or only insufficient feedback from the other end-point, a retransmission timer is the only way to recover a lost packet [Zha86]. We explicitly focus on the design and the performance of retransmission timers intended for reliable unicast transmission across the Internet. Hence, alternative design approaches that do not require a retransmission timer, e.g., those based on data-carousels [BLM02], are outside the scope of this paper.

The two reliable unicast transport protocols standardized by the Internet Engineering Task Force (IETF) today are the Transmission Control Protocol (TCP) [RFC793], and the Stream Control Transmission Protocol (SCTP) [RFC2960]. Although, some slight adjustments have been made for SCTP, both protocols basically use the retransmission timer devised by Jacobson [Jac88] that was later adopted as a standard [RFC2988].

A number of problems are known related to the algorithm for calculating the retransmission timeout value (RTO) as defined in [RFC2988]. We refer to this algorithm as the RFC2988-RTO. One problem is the interaction of the RFC2988-RTO with the Round-Trip Time Measurement (RTTM) mechanism that is based on the TCP Timestamps option [RFC1323]. With the RTTM mechanism the sender times every packet instead of only one packet per RTT. This interaction often causes the RTO to follow the RTT too closely. Other problems of the FRC2988-RTO that exist with and without using timestamps are its sluggish response to delay spikes, and its rather conservative response to sharp drops of the RTT.

Addressing those problems has been the motivation for the work presented in this paper. We propose a new algorithm for calculating the RTO, named the "Peak-Hopper"-RTO (PH-RTO), that removes the mentioned interaction with the RTTM mechanism, and also solves the other known problems with the current RTO algorithm. We have not identified any problems with the rules concerning how Round-Trip Time (RTT) samples are taken, the timer granularity (G), or the management of the retransmission timer (see Section 3-5 in [RFC2988]). Hence, the Peak-Hopper retransmission timer (PH-Timer) only introduces a new RTO algorithm, the PH-RTO, but otherwise complies to [RFC2988].

Based on simulations in ns2 [NS] we compare the performance of both the RFC2988- and the PH-Timer in various scenarios. Previous work has shown that the performance of the RFC2988-Timer is dominated by the RTO minimum of 1 second [AP99]. To have a fair comparison, we have therefore replaced that minimum in our simulations with "(RTT-Sample + 2 * G)" as used in the FreeBSD implementation of TCP. Hence, throughout the rest of the paper, our definition of the RFC2988-RTO/-Timer is identical to the specification in [RFC2988] except for the RTO minimum.

Our results show that in most cases the PH-Timer yields better performance both in terms of quick loss detection when a packet was lost, and in terms of minimizing the number of spurious timeouts. The PH-Timer on average detects a lost packet up to one RTT quicker than the RFC2988-Timer while at the same time causing less or at least not more spurious timeouts. Moreover, the loss detection times for the PH-Timer are much more closely spread around the average whereas the RFC2988-RTO often generates "RTO outliers" that lead to exceptionally long loss detection times. This highlights a major weakness of the RFC2988-Timer in certain scenarios: during some phases of a TCP connection's lifetime it is often too aggressive while in other phases it is often too conservative. I.e., in those scenarios the "relative distance" between the RTT and the RTO computed according to the RFC2988-RTO often varies massively while for the PH-Timer this "relative distance" is rather constant.

We further demonstrate the benefit of using the RTTM mechanism. In most scenarios a considerably better retransmission timer performance is achieved when timing every packet instead of only one packet per RTT. This is especially true when the congestion window is large and many packets (RTT samples) are in flight.

It is well known that a retransmission timer is really only a last resort, and that much better TCP performance can be achieved with advanced loss recovery schemes based on feedback from the other end-point [Zha86]. This has been the motivation for many of the advanced loss recovery schemes (being) standardized for TCP: fast retransmit [Jac88][RFC2581], NewReno [RFC2582], limited transmit [RFC3042], SACK-based loss recovery [RFC3517], and early retransmit [AAAB03]. For bulk data transfers, those schemes have made TCP's performance rather independent of the retransmission timer. For such connections the retransmission timer is merely needed in certain corner cases, e.g., when the transfer's last packet is lost. We were able to validate that in our simulations: with all these schemes turned on, it has become rather difficult to generate timeout events! For interactive applications, however, TCP's performance to a large extent still relies on the retransmission timer. This is because with such connections often too few or no acknowledgements (ACKs) are in flight to trigger any advanced loss recovery scheme.

The paper is organized as follows. In Section 2, we review known problems of the RFC2988-RTO. In Section 3, we introduce the PH-Timer where we first motivate our design decisions, and then explain its features in detail. In Section 4, we present our analysis methodology including the scenarios we used in our simulations, and the metrics we used for evaluating the performance of a retransmission timer. In Section 5, we present and discuss our simulation results. In the final Section 6, we conclude the paper, point out remaining open issues, and suggest future work.

II. Discussing the RFC2988-Timer

In this section, we discuss relevant issues of the RFC2988-Timer. In the first sub-section, we address the 1 second minimum defined for the RTO in [RFC2988]. In the second sub-section, we discuss a feature of the RFC2988-Timer that results from the rules that govern the management of the retransmission timer. In the following three sub-sections, we discuss known problems of the RFC2988-RTO. Since we refer to the specification of the RFC2988-RTO at a number of places, we repeat it below.

For every (except the first) RTT measurement (RTT-Sample) the RTO is computed as follows:

RTTVAR ← 3/4 * RTTVAR + 1/4 * |SRTT - RTT-Sample|

SRTT ← 7/8 * SRTT + 1/8 * RTT-Sample

Eita implemented.

RTO ← SRTT + max (G, 4*RTTVAR)

where RTTVAR denotes the RTT variation, and SRTT denotes the smoothed RTT.

A. The RTO Minimum

spurious mane fake

The RTO minimum should be seen as a necessity to protect against spurious timeouts in situations where the RTT is close to or even below the timer granularity (G), or when the RTO converges to an RTT that does not vary much.. In all other cases, the minimum should only have little effect. Otherwise, the RTO has failed as a predictor of an appropriate upper bound for the RTT. When using a heartbeat timer, the RTO minimum must at least be 2*G [WS95]. The original proposal of the RFC2988-RTO [Jac88] has used an RTO minimum of 2*G with a G of 500 ms. For the standard [RFC2988] the minimum has

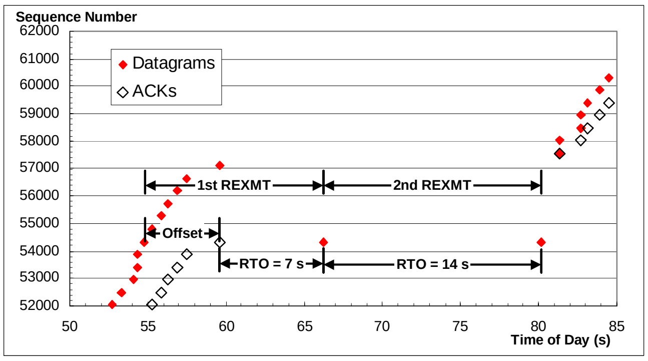


Figure 1. The Age of the Oldest Oustanding Segment

been changed to a fixed (!) value of 1 second. The fact that the RFC2988-RTO's performance is dominated by that minimum [AP99] can be viewed as a strong indication that that RTO algorithm would have difficulties without such a conservative minimum. In fact, that is what our results confirm as explained in more detail in Section 5. Recall from Section 1, that our definition of the RFC2988-RTO/-Timer is identical to the specification in [RFC2988] except for the RTO minimum. We have replaced it with "(RTT-Sample + 2 * G)" as used in the FreeBSD implementation of TCP.

Since RTTs commonly found in the Internet are in the range of 50-500 milliseconds, a minimum of 1 second may seem acceptable, but for connections with much lower RTTs this may not be acceptable. We believe that any new proposal for an RTO algorithm needs to perform well without a fixed minimum.

B. The Age of the Oldest Outstanding Segment

According to the timer management specified in [RFC2988], the retransmission timer is restarted for every acceptable ACK. For bulk data transfers this implicitly adds a safety margin of roughly the latest RTT sample to the loss detection time. This is denoted as "offset" in Figure 1. The retransmission timer will expire for the first time after a time that is the sum of the "age of the oldest outstanding segment" and the RTO.

In previous work, this had been identified as a "bug" [LS99]. But, our analysis has convinced us that this is actually a feature. This "responsive safety margin" compensates for the problems explained in the following two sub-sections, and especially for the sluggish response to delay spikes often avoids spurious timeouts. This can also be seen in our results presented in Section 5. It important to note, though, that this safety margin does not exist for highly interactive applications where often only a single packet is in flight.

Also the PH-Timer benefits from this feature since it is based on the same rules for managing the retransmission timer as the RFC2988-Timer.

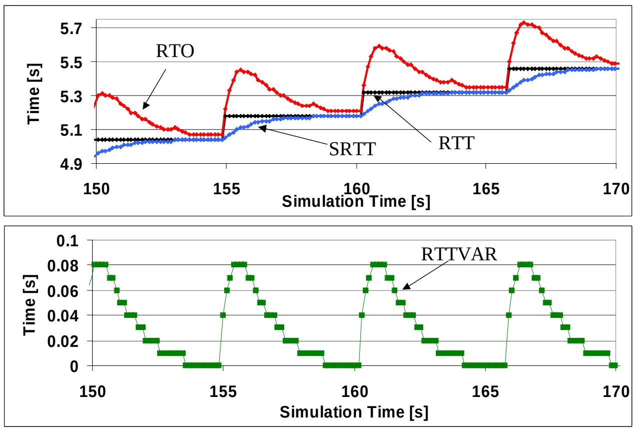


Figure 2. The Gains Don't Scale

It is important to note that in all plots in this paper that show a graph of RTO, this offset is not considered, i.e., we always only plot the "true" value of RTO.

C. The Gains Don't Scale

Every RTT-Sample is factored into the algorithm with a gain of 1/8 and 1/4, respectively, while the algorithm keeps a memory of the connection's lifetime with a gain of 7/8 and 3/4, respectively. An illustrative way of thinking of this is to say that the SRTT averages over the last 8 RTT samples while the RTTVAR averages over the last 4 RTT samples. Those constants have been chosen under the assumption that the TCP sender only times one packet per RTT [Jac88]. However, if the sender times every packet, the fixed constants often lead to an RTO that is too aggressive. The problem becomes worse as the congestion window increases, and more RTT samples return per RTT. What happens is that the sender "forgets too fast", i.e., the memory that the algorithm maintains is truncated, and the RTO merely becomes a function of the most recent RTT samples. This leads to an RTO that converges to the RTT too fast as shown in Figure 2. In that scenario a single TCP connection runs across a bottleneck with a high bandwidth-delay product, and every packet is timed. In that case, the RTT samples form a step function as soon as packets start queueing at the bottleneck. As a result, the SRTT quickly converges to the RTT while the RTTVAR quickly converges to zero.

This interaction of the RFC2988-RTO with the timing of every packet, e.g., by using the TCP Timestamps option and the RTTM mechanism [RFC1323], was shown in previous work [LS99]. The problem is also known in the IETF community (see Section 3 in [RFC2988]), and is keeping [RFC1323] and [RFC2988] from advancing on the IETF's standards track.

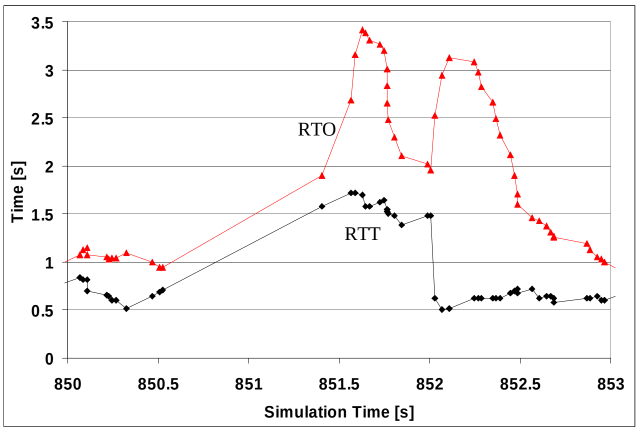


Figure 3. Sluggish Response to Delay-Spikes

D. Sluggish Response to Delay-Spikes

The RFC2988-RTO is sluggish in responding to a sudden and large increase in the RTT, a so-called delay spike. This is shown in Figure 3. Between seconds 850.5 and 851.5, the RTT suddenly increases by one second. However, due to the offset described in Section 2.2, this delay spike does not lead to a spurious timeout. The important point to note is that despite the rather large delay spike, the resulting RTO computed shortly before second 851.5 lies just above the RTT. In fact, that RTO is the minimum of RTT-Sample + 2*G. It takes another 3 RTT-Samples to raise the RTO to an appropriate level of about 3.3 seconds. Note that every packet is timed in this case. This sluggishness of RTO algorithms that are based on low pass filters - (1-beta) * "history" + beta * "present" - has already been observed 20 years ago [RFC889], [Zha86]. This problem is even more pronounced without the use of timestamps since it will take the RFC2988-RTO longer (in time) to adapt to the appropriate level.

Note that the problem discussed in the previous sub-section is also visible in Figure 3. It can be seen in the steep decline of the RTO between seconds 851.5 and 852. Since the RTT remains relatively stable during that time the SRTT again converges to the RTT too soon while the RTTVAR too quickly converges to zero.

E. The Prediction Flaw

Another problem of the RFC2988-RTO can be seen in Figure 3 at second 852. At that time the RTT drops back to the level it had before the delay spike. In response to the drop, the RTO shoots up to a level 6 times higher than the RTT! The problem is that the RTTVAR does not distinguish between positive and negative variations, and only responds to absolute changes in the RTT. Thus, the RFC2988-RTO's response to a sharp RTT drop is the same as the response to a delay spike. As opposed to the problems explained in the previous two sub-sections, this problem will lead to an RTO that is too conservative. This problem is more pronounced with the use of timestamps. The problem has been explained in previous work [LS99].

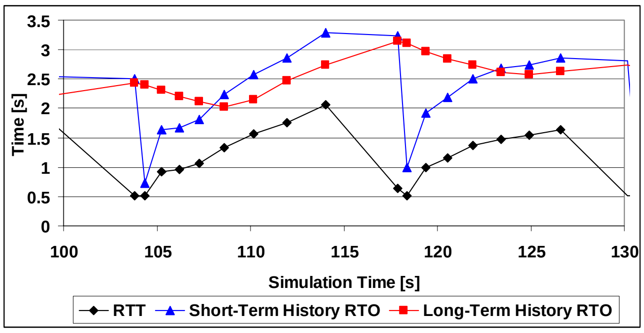


Figure 4. The PH-RTO is calculated as the envelope of the Short- and Long-Term History RTO curves

III. THE PEAK-HOPPER

A. The Basic Idea

We have adopted two key ideas from previous work when designing the Peak-Hopper-RTO. First of all, in [RFC889] Mills suggests that the RTO should be responsive to upward-going trends in the RTT and less responsive to downward-going trends. Secondly, in [Jac88] Jacobson introduces the notion of making the RTO dependent on the RTT variance. While the PH-RTO realizes the variance measurements in a different way, the fundamental idea remains the same.

The main novelty introduced by the Peak-Hopper is that it essentially runs two RTO algorithms in parallel. One algorithm (denoted Short-Term History RTO in Figure 4) monitors the present and short-term history in order to respond to RTT increases. The other algorithm (denoted Long-Term History RTO in Figure 4) simply decays the current value of RTO, and can therefore be said to represent the long-term history. The PH-RTO is then set to the maximum of the two resulting RTO values. Another novelty of the PH-RTO is that if the first RTO algorithm detects an RTT increase, the PH-RTO resets its long-term history and completely orients its RTO calculation on the short-term memory. In this sense, we have taken Mills' suggestion to the extreme. One of the benefits of this approach is that the performance of the PH-RTO is not affected by packet length effects [RFC889], [Zha86].

B. Calculating the PH-RTO

We describe the details of how the PH-RTO is calculated in six steps.

$$\delta = \frac{RTT_{sample} - RTT_{previous}}{RTT_{previous}} \quad \text{(Step 1)}$$

$$D = 1 - \frac{1}{F * S} \quad \text{(Step 2)}$$

$$B \leftarrow \max(\delta, D * B) \quad \text{(Step 3)}$$

$$RTT_{\max} = \max(RTT_{sample}, RTT_{previous}) \quad \text{(Step 4)}$$

$$RTO \leftarrow \max(D * RTO, (1+B) * RTT_{\max}) \quad \text{(Step 5)}$$

$$RTO \leftarrow \max(RTO, RTO_{\min}) \quad \text{(Step 6)}$$

Having collected a new RTT sample, RTT_{sample}, we compare this value to the previous RTT sample collected, $RTT_{previous}$, as shown in Step 1. We call the normalized change between these two samples δ. This is the measure of the short-term changes in RTT. The rationale for monitoring this change is that if δ is large enough (i.e. if there has been a large increase in RTT), then the RTO should be increased.

In Step 2, we further define a decay factor, D. As will become apparent, D determines how rapidly the RTO is decayed. We also introduce a fader variable, F, which controls the speed of this decay (a high F gives a slow decay and vice versa). F is greater or equal to 1. Later in this section we discuss the configuration of F.

S is used to denote the number of RTT samples that are expected to be collected from the packets currently in flight. If the timestamps option is enabled, S is set to the number of packets in flight (or *flight*). Provided no packets in the flight are lost and no DUPACKs are received this corresponds to the number of RTT samples collected. Note that in this case, S is time varying since *flight* varies with time. If the timestamp option is not used only one RTT sample is taken per RTT, hence S is set to 1[1]. Note that since F is greater than 1, D is always smaller than 1.

In Step 3, we calculate a booster variable B. The booster variable determines how high the RTO should "hop" when a large RTT increase has been detected. We set B to the maximum of δ (calculated in Step 1) and the decayed current value of B. Note that B is greater than 0.

In Step 4, we set RTT_{max} to the maximum of the new RTT sample, RTT_{sample}, and the previous RTT sample, $RTT_{previous}$. RTT_{max} is used to represent the short-term history of the RTT.

Step 5 is the most central in the PH-RTO calculation and realizes the basic idea of the PH-timer as previously described. We set RTO to the maximum of a long-term history (represented by the term D*RTO) and the short-term history (represented by the term (1+B)*RTT_{max}). If the long-term history dominates, D determines how quickly the RTO is decayed. If the short-term history dominates (i.e. if an RTT peak has been observed), B dictates how high the RTO should hop. This process is depicted in Figure 4.

The rationale for erasing the long-term history in this manner is simple: if a large increase in RTT (relative to the long-term history) is observed, we should forget what happened in the past and orient the new RTO setting completely to this new RTT peak.

In a final step (Step 6), we ensure that the RTO does not fall below the minimum allowed RTO, RTO_{min}. We discuss the setting of RTO_{min} in Section D.

The PH-RTO only has a single parameter to configure: the fader variable F. As previously mentioned, F determines how fast the RTO and B decay in the absence of a a large RTT increase. As such, the parameter determines the conservativeness of the PH-RTO.

We have experimented with different values of F, and found that when using timestamps, F = 16 gave a good tradeoff between promptness and accuracy. This value was used in our simulations. We also found that when not using timestamps, a slightly higher value of F is required. Since only one packet per flight is being timed without timestamps, the RTO calculation is based on more incomplete information from the network. This motivates a slightly more conservative RTO calculation. We therefore settled for F = 24 when not using timestamps in our simulations.

C. *Initializing the RTO*

The initialization of the PH-RTO differs slightly from the initialization of the RFC2988-RTO. We changed the initialization in order to minimize the effect of the "packet length effect" [Zha86][RFC3481], which may lead to an inaccurate RTO calculation on low bandwidth paths. In Figure 5 we illustrate the initialization process and the terms we use in the following discussion.

At time t_0, we set

RTO = 3 seconds

and send the initial SYN segment. The SYN ACK is received RTT_{SYN} seconds later, at t_1. At this point, we set

$$RTO = 3 * RTT_{SYN},$$

and send the first data segment. So far, the initialization is identical to what is defined for the RFC2988-Timer. For bandwidth-dominated paths, where the RTT of a packet is proportional to the packet length,

[1] Note that it is possible to get RTT samples from every ACK although the timestamp option is disabled [SK02]. In the simulations and discussion in this paper, however, we assume one RTT sample per *flight* when no timestamps are used.

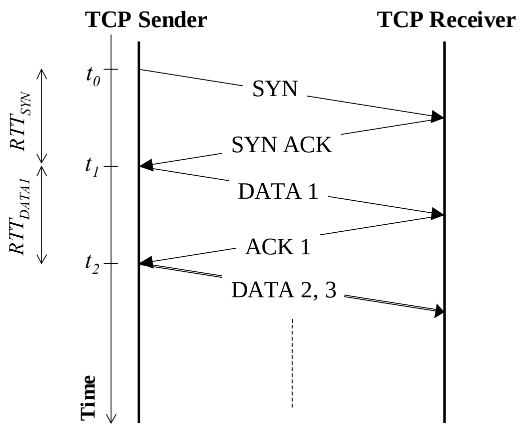


Figure 5. Setup of a TCP Connection

the RTT measurement of the first (full sized) data segment will be considerably larger than that of the relatively small SYN segment. Hence, for such paths we can expect $RTT_{DATA1} >> RTT_{SYN}$. This motivates the seemingly conservative setting of the RTO for the first data segment above.

At time t_2 upon receiving the ACK for the first data segment, we re-initialize the RTO as follows:

$$RTO = (1 + B_{INIT}) * RTT_{DATA1},$$

where B_{INIT} is the initial value of the Booster and RTT_{DATA1} is the RTT measurement of the first data segment. RTT_{DATA1} is the first RTT measurement of a full-sized segment, and especially for bandwidth-dominated paths, this measurement serves as a better basis upon which to make predictions on future RTT samples than RTT_{SYN}. From this point on, the RTO is calculated as already described in Section B above.

In our simulations, we have set B_{INIT} to 0.75.

D. *The RTO Minimum*

We define the minimum RTO-value, RTO_{min}, to

$$RTO_{\min} = RTT_{\max} + 2 * G,$$

where RTT_{max} is the maximum of the two last RTT measurements (as discussed in Section B) and G is the granularity of the heartbeat timer of the operating system at the TCP sender. In simulations, we have seen that using RTT_{max} to calculate the minimum RTO can greatly reduce the number of spurious retransmissions. The second term of the equation above, 2*G, adds a safety margin to RTO_{min}. As mentioned in Section I, this term is needs to be added when the retransmission timer is used in combination with a heartbeat timer.

IV. Analysis Methodology

We compared the performance of PH-Timer with the RFC2988-Timer through simulations using version 2.1b9a of the ns-2 simulator [NS]. For the evaluation, we have defined a number of metrics, with which we believe to fairly assess the performance of a retransmission timer. These are described in this section as well as the simulation scenarios for which the retransmission timers were evaluated. However, first we describe how the TCP protocol was configured in the simulations.

A. *TCP Configuration*

As previously discussed, a TCP equipped with enhancements such as SACK, fast retransmit, limited transmit and ECN [RFC3168] will be able to complete most bulk data transfers without being forced to use its retransmission timer for loss recovery. This was also confirmed during our simulation work, where we indeed had problems to produce timeout events with the above-mentioned TCP enhancements enabled.

In order to see the effects of the different retransmission timers, we had to somehow make the TCP-performance more dependent on its retransmission timer. We therefore decided to disable the above-mentioned TCP enhancements and instead simulate a very primitive TCP-sender. The only loss recovery mechanism of this primitive TCP sender is its retransmission timer. Hence, all packet losses were recovered by means of a timeout-based retransmission.

We do not claim that such primitive TCP-senders are found anywhere in real-life, but we found this to be an effective way to collect sufficient statistics to enable a meaningful performance analysis of the different retransmission timers at hand.

In the simulations, we have used a heartbeat timer granularity, $G = 10$ms.

B. *Simulation Scenarios*

Our aim was to evaluate the performance of the retransmission timers in scenarios that exhibit a wide range of different RTT patterns. We have defined four such scenarios, each described in detail below.

1) Scenario A – High Degree of Statistical Multiplexing

In this scenario, multiple flows share a bottleneck link, as depicted in Figure 6. Each of the N senders (S_i, $i \in \{1…N\}$), communicates with one receiver (R_i, $i \in \{1…N\}$). All links from the senders, S_i to Router1 as well as all links from Router2 to the receivers, R_i, have a data-rate of 3Mbps and a latency of 50ms. The bottleneck link, L_B, has a data-rate of 1.5 Mbps and a latency of 50ms. The bottleneck link queue, Q_B, is a passive queue with buffer-space for 100 IP-packets. The queue applies a drop-tail dropping policy.

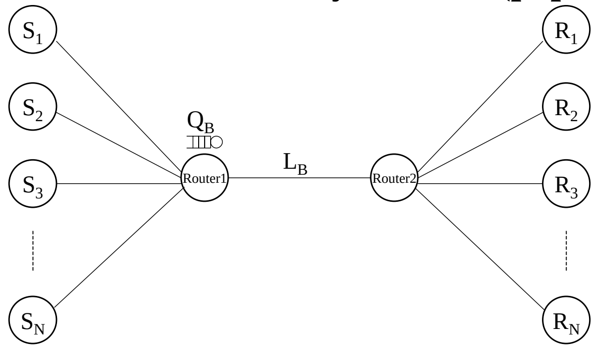


Figure 6. Simulation Topology used in Scenario A

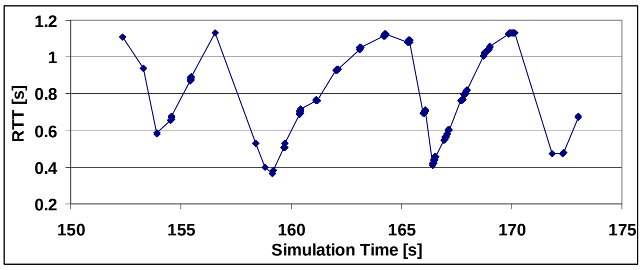


Figure 7. Typical RTT-pattern for Scenario A

Each sender transmits an FTP-file consisting of exactly 200 IP packets to its receiver. The start time of the transfers were uniformly distributed in the interval (0, 1000) seconds. A typical RTT pattern arising from this scenario is shown in Figure 7. The drastic RTT variations in the figure arise from global synchronization effects: when the passive queue overflows, many (or all) senders are forced into a timeout and thereby drastically reduce their load. The queue size then decreases and with that also the RTT.

2) Scenario B – Modem Connection

We simulate a single sender, S_1, which communicates with a single receiver, R_1 as shown in Figure 8. The bottleneck link, L_B, has a data-rate of R_B kbps (R_B assumed various values during the simulations) and a one-way latency of 50 ms. The bottleneck link queue had buffer-space for approximately one pipe-capacity and employed passive queuing with a drop-from-front dropping policy. Both links L_1 and L_2 have a latency of 50 ms.

The main characteristic of this simulation scenario is that the RTT at the TCP sender is determined by the amount of queuing delay added at Q_B as shown in Figure 9. This amount is in turn directly proportional to the load of the TCP sender.

3) Scenario C – WCDMA Link

This simulation scenario is very similar to Scenario B. However, we have replaced the link-layer of L_B in Figure 8 with a WCDMA-link model implemented for the ns-2 simulator. The WCDMA-link model is based on the

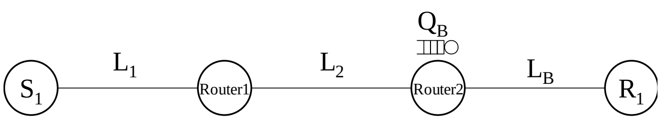


Figure 8. Simulation Topology used in Scenarios B and C

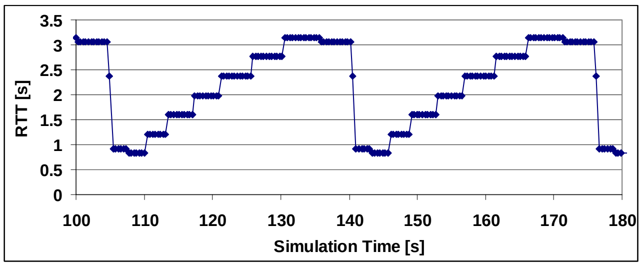


Figure 9. Typical RTT-pattern for Scenario B

results in [PM01]. When calculating the transmission delay of an IP-packet, this model takes into account block-errors and the corresponding retransmissions on link-layer. The resulting RTT seen by the TCP sender is therefore determined by the queuing delay at Q_B (like in Scenario B), but additionally by the variable transmission delay arising from the modeled link-layer retransmission. A typical RTT pattern for this scenario is shown in Figure 10.

4) Scenario D – Delay Spikes

The bottleneck link, L_B, used in this scenario was the same as in Scenario C, i.e. models a WCDMA-link. However, the capacity of the bottleneck link queue, Q_B, was larger than the advertised window of the receiver R_1. Hence, no packets were dropped at Q_B. The main difference in this scenario to the previous ones is that the bottleneck link L_B was interrupted in downlink direction (from Router2 to R_1) at regular intervals. These types of link interruptions may occur during mobility procedures in wireless networks [RFC3481]. The obvious effect was that packets were queued for longer times at Q_B, and the TCP sender observed sudden increases in the measured RTT. The bottleneck link was interrupted with a periodicity P_B and every interrupt had a duration D, where P_B and D were constant per simulation run. All links L_1, L_2 and L_B have a latency of 50 ms. We simulated two different rates for the bottleneck link, L_B: 64 and 384 kbps. The topology for the scenario simulated

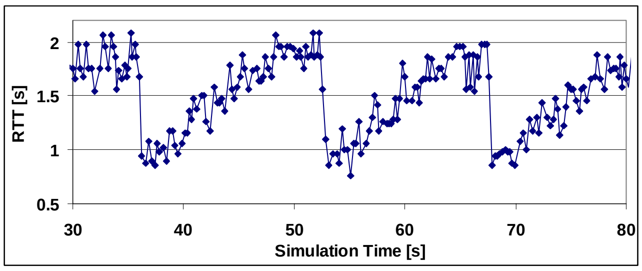


Figure 10. Typical RTT-pattern for Scenario C

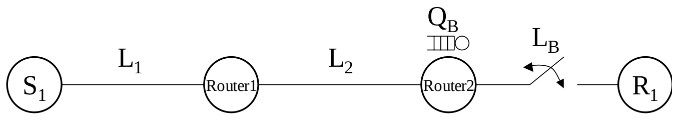


Figure 11. Simulation Topology used in Scenario D

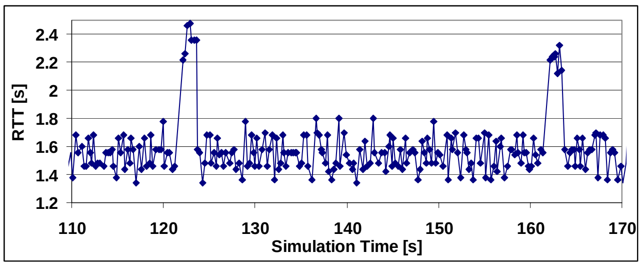


Figure 12. Typical RTT-pattern for Scenario D

is shown in Figure 11. A typical RTT-pattern for this scenario is shown in Figure 12, where P_B = 40 seconds and D = 1 second.

C. *Performance Metrics*

The ideal retransmission timer is both accurate (i.e. avoids spurious timeouts) and prompt (i.e. retransmits without introducing excess delays when necessary). In order to assess the performance of the retransmission timers we have introduced a number of metrics, some of which have been reused from [AP99].

1) *Accuracy*

In order to assess the accuracy of the retransmission timer, we measure the relative amount of unnecessary traffic injected onto the network by the retransmission timer as

$$B_{norm} = \frac{\#SpuriousTimeouts}{\#DataSegments}.$$

In the equation, *#SpuriousTimeouts* is the total number of spurious timeouts observed during a simulation, and *#DataSegments* is the total number of data segments injected into the network by all senders during a simulation.

2) *Waiting-Time*

We call a timeout "necessary" if no 3rd DUPACK has arrived at the sender by the time that an acceptable ACK for the retransmitted segment arrives at the sender.

In order to assess the promptness of the retransmission timer, we define the normalized waiting time for the *i*:th necessary timeout in a simulation run as

$$W_i = \frac{RTO_i}{RTT_i}.$$

RTO_i is the value with which the retransmission timer was started for the retransmitted segment, and RTT_i is the last RTT measurement before the timeout. We calculate the mean normalized waiting time for a necessary timeout as

$$W = \frac{1}{M}\sum_{i=1}^{M} W_i.$$

M is the total number of necessary timeouts in a particular simulation run. Expressed in words, *W* measures the mean time that the retransmission timer waited until it expired, thereby producing a necessary timeout, expressed in multiples of the last RTT measurement prior to the timeout.

We define $W_{95\%}$ as the value for which 95% of all W_i values collected in a simulation run are smaller than $W_{95\%}$. This metric reveals how long the waiting time is for the retransmissions that had to wait the long for the retransmission timer to expire. In effect, this is a predictability metric: a high $W_{95\%}$ reveals that the retransmission timer in some cases waits excessively long to retransmit, while a low $W_{95\%}$ suggests that the retransmission timer made all its necessary retransmissions in a narrow spread of waiting time.

3) *Other Metrics*

For simulation runs where none or only a few timeout-events were observed, we cannot fairly judge the performance of the retransmission timers based on the accuracy and promptness metrics introduced above. Hence, we need another metric to judge the performance of the retransmission timer.

In this case, we chose use the proximity of the RTO to the RTT as a performance metric. This was calculated as RTO_N/RTT_L where RTO_N is the newly calculated RTO and RTT_L is the last RTT sample prior to the RTO update. We then calculated *P* as the mean value of this fraction over an entire simulation run.

Obviously, when comparing the performance of Retransmission Timer A with Retransmission Timer B for a certain simulation run, Retransmission Timer A can be said to have performed better than Retransmission Timer B if its P is lower (i.e. the RTO was in mean closer to the RTT), provided none of the timers caused any (or only a few) timeouts during that simulation.

In fact, the W_i samples can be said to be a subset of the P samples. Using *P*, the RTO calculation is continuously sampled (every time the RTO is updated), while W_i is only sampled at specific events (when a necessary timeout occurs).

	W		$W_{95\%}$		Bnorm	
Flows	RFC2988	PH	RFC2988	PH	RFC2988	PH
700	1.7	1.3	2.5	1.7	19.1	17.1
650	1.5	1.3	2.0	1.7	8.5	7.4
600	1.5	1.4	2.1	1.7	0.2	0.4
550	1.5	1.5	2.0	1.7	0.3	0.1
500	1.6	1.6	3.0	1.7	0.0	0.0

Table 1. W, W95% and Bnorm results for simulations with timestamps

	W		$W_{95\%}$		Bnorm	
Flows	RFC2988	PH	RFC2988	PH	RFC2988	PH
700	2.4	2.3	4.5	4.2	18.8	19.4
650	2.4	1.8	3.2	2.9	4.2	6.0
600	2.0	1.9	3.0	2.5	0.0	0.0
550	2.0	1.7	3.0	2.5	0.0	0.2
500	3.3	1.9	4.7	3.0	0.0	0.0

Table 2. W, W95% and Bnorm results for simulations without timestamps

V. RESULTS

A. *Benchmarking: PH-Timer vs. RFC 2988-Timer*

1) *Scenario A – High Degree of Statistical Multiplexing*

In this scenario, we varied the number of senders sharing the bottleneck link in steps of 50 from 700 to 500 users. With this variation, the packet loss rate varied from approximately 15% in the simulations with the highest number of senders to fractions of a percent in the simulations with the lowest number of senders.

a) *Results With Timestamps*

Table 1 shows the results of the simulations that were carried out with TCP senders using timestamps. We see that the PH-Timer overall performed better in all metrics. The PH-Timer overall achieved a lower waiting time, e.g. with a 24% lower W for 700 flows. Furthermore, the RFC2988-Timer suffers from a heavy-tailed W distribution, as revealed by the $W_{95\%}$ metric. For example, for 5% of the necessary timeouts in the case of 700 flows, the RFC2988-Timer waited for more than 2.5 RTTs before it expired. The corresponding figure for the PH-Timer is 1.7 RTTs. We also see that the PH-Timer was more accurate, i.e. produced less spurious retransmissions, than the RFC2988-Timer with a slightly lower B_{norm} in most cases.

b) *Results Without Timestamps*

Table 2 shows the results of the simulations that were carried out with TCP senders not using timestamps. In this case, the PH-Timer also performed considerably better in terms of the waiting-time W, giving an improvement in

W of up to 40%. Similar improvements can be observed when looking at $W_{95\%}$. This heavier tail of the W distribution for the RFC2988-Timer reveals that the loss recovery provided by the RFC2988-Timer was more unpredictable and had more retransmissions with a high waiting time than the PH-Timer. However, without timestamps, the PH-Timer overall performed slightly worse than the RFC2988-Timer in terms of accuracy (B_{norm}).

When comparing the performance of the retransmission timers using timestamps (see Table 1) with the case when no timestamps are used (see Table 2), it becomes apparent that timing every segment improves the timer performance in this scenario. Although there is no significant improvement in the accuracy (B_{norm}), the waiting time W and $W_{95\%}$ are decreased considerably leading to a prompter loss recovery.

2) *Scenario B – Modem Connection*

In this scenario, neither of the retransmission timers produced any spurious timeouts. This was expected, since the generated RTT samples were too predictable for this to happen.

Furthermore, the necessary timeouts all occurred at the end of a congestion control cycle. Hence, for a specific timer, the waiting-time, W, for all of these timeouts was approximately constant, since the RTT history prior to the timeout was the same in all cases. We therefore do not believe that the metric W is representative of the timer performance in this scenario, since it only gives information about the value of the RTO at the end of the congestion control cycle. Instead, we base the evaluation on the RTO-proximity metric, P, from which we can extract more information.

From Figure 13 we see that the PH-Timer performs considerably better in terms of P than the RFC2988-Timer both with and without timestamps. We interpret this result as follows: if (for any reason) the TCP sender has to rely on its retransmission timer for loss recovery, the PH-Timer would in mean recover the lost packet in mean up to up to 40% faster than the RFC2988-Timer. This is true both for the case with and without timestamps.

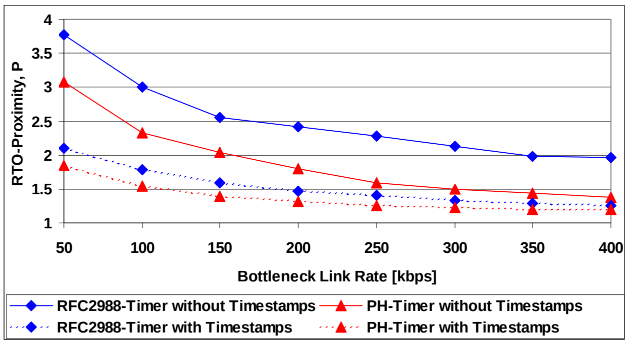


Figure 13. Mean RTO-Proximity values for Scenario B

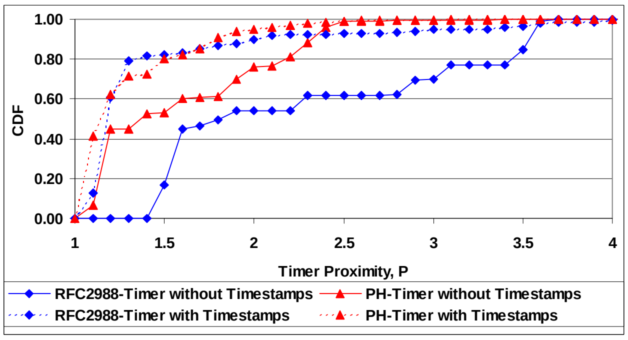


Figure 14. RTO-Proximity distribution with and without timestamps

Figure 14 shows the distribution of the individual P values for both timers with and without timestamps. We see that without timestamps, the RFC2988 timer is generally much more conservative than the PH-Timer with no P-values smaller than 1.4. Also, even in this well-defined scenario, the RFC2988-Timer has P values as large as 3.5. This is rather conservative given the predictable nature of the underlying RTT samples (see Figure 9).

The most notable characteristic of the RFC2988-Timer distribution with timestamps is the heavy tail of the distribution: 8% of the P-values are greater than 2.5. The corresponding value for the PH-Timer is 1%. This is an effect of the previously described "Prediction Flaw" whereby the RFC2988-RTO becomes overly conservative in response to a drop in the measured RTT. Such drops occurred frequently following a packet loss at the bottleneck link queue (see Figure 9). This flaw makes the performance of the RFC2988-Timer very unpredictable: in some cases it would expire rapidly, while in others it would introduce excessive waiting times. We note that the PH-Timer does not suffer from any such problems: its P-distribution with timestamps has less density for very high and very low P values.

Figure 13 and Figure 14 also reveal that the performance of both timers is improved when using timestamps: timer proximity, *P*, is decreased by up to 40% when using timestamps.

3) Scenario C – WCMDA link

For the same reason as in the pervious scenario, we base our evaluation on the collected P values. We see from Table 3 that the PH-Timer outperforms the RFC2988-Timer in all simulated scenarios in terms of the timer proximity. In some cases the difference is as large as 13%. Again, in cases when a timeout would have been necessary, this would have speeded up the loss recovery.

We do not show a P-distribution plot at this point, since the main characteristics are the same as was already shown in Figure 14. However, we note that the problem

	With Timestamps		Without Timestamps	
Rate	RFC2988	PH	RFC2988	PH
384	1.97	1.85	2.91	2.54
64	2.03	1.92	3.38	2.94

Table 3. Mean RTO-Proximity values for Scenario C

of the heavy tail of the RFC2988-Timer when using timestamps is also present in this scenario. This is again caused by the "Prediction Flaw" that makes the performance of the RFC2988-Timer highly unpredictable.

Again, we note that the performance of both retransmission timers is improved when using timestamps, i.e. the RTO-proximity is decreased considerably.

4) Scenario D – Deterministic Delay Spikes

In this scenario, we are interested in evaluating the robustness of the timers to delay spikes. As described in Section 4, interrupting a link from the sender to the receiver generated the delay spikes. As a performance metric, we calculated the percentage of link interruption events that generated a spurious timeout (since this metric is specific to this scenario, it was not introduced in Section IV). In Figure 15, this percentage is plotted as a function of the duration of link interruption.

We see that for both simulated bottleneck link rates and independent of whether timestamps are used, the PH-Timer is more robust to delay-spikes than the RFC2988-Timer. In some cases, the PH-Timer is up to 40% less likely to cause a spurious timeout.

The reason for the increased robustness of both timers to delay-spikes with the lower rate (64kbps) is that the baseline delay is higher for this scenario due to the higher transmission delay. Thus, a delay spike of constant length is smaller relative to the baseline RTT, and therefore easier to withstand.

Figure 16 depicts typical traces from this experiment for both timers with and without timestamps. All four plots display the RTT (lower graph), and the RTO (upper graph) that was calculated by the timer from this RTT. As can be seen in the individual plots, the link was interrupted at 200s for 0.8 seconds, causing a delay-spike. By visual inspection we have determined the half-life, i.e. the time it takes for the RTO to decay from it's maximum value ensuing from the delay-spike to half of that maximum, for each plots.

We found that the behavior of the PH-Timer is largely independent of the use of timestamps: the increase ensuing from the delay-spike is comparable (3.1 seconds with timestamps and 2.5 seconds without timestamps), while the half-life of the RTO without timestamp (10 seconds in Figure 16 a) is comparable to that with timestamps (6.9 seconds in Figure 16 c). In fact, the more

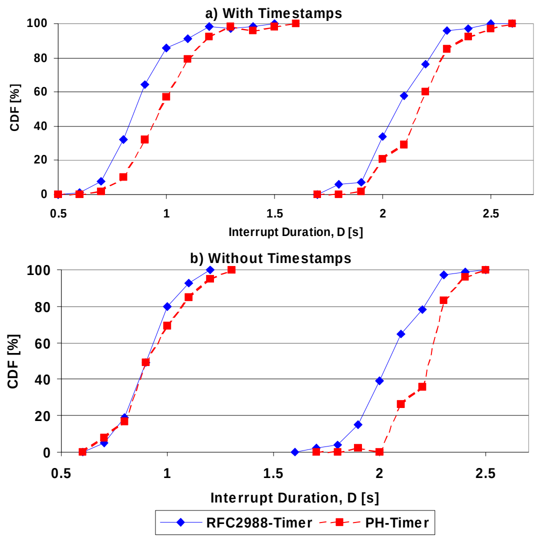


Figure 15. Percentage of Link Interruptions that caused a Spurious timeout as a function of the Interruption Duration.

conservative decay of the timer without timestamps is a conscious design choice: we have chosen a more conservative fader variable, *F*, for this case as already discussed in Section III.

The characteristics of the RFC2988-RTO, on the other hand, changes completely when using timestamps. First of all, the maximum RTO value ensuing from the delay-spike differs greatly: it is over 3 seconds with timestamps (Figure 16 d) while the maximum stays at a mere 1.5 seconds without timestamps (Figure 16 b). The second difference is the half-life: the half-life without timestamps is very long (14.4 seconds), while it is extremely short when using timestamps (0.4 seconds). The reason for this rapid decay is ”The Gains Don’t Scale”-Flaw described in Section II.

This shows that the PH-Timer is able to eliminate the interactions with the RTTM mechanism defined for the TCP timestamps option. The performance of the PH-RTO is largely invariant to whether every packet or only one packet per flight is timed.

VI. CONCLUSIONS

We have evaluated the performance of the retransmission timer that is standardized for TCP and SCTP [RFC2988], and widely deployed in the Internet. Through extensive simulations in ns2 that covered a number of common scenarios, we have demonstrated that problems related to the RTO algorithm impede the performance of the RFC2988-Timer. We provided a detailed analysis of these problems. Although most of the problems had been mentioned before in various previous studies, this paper - to our knowledge - provides the first

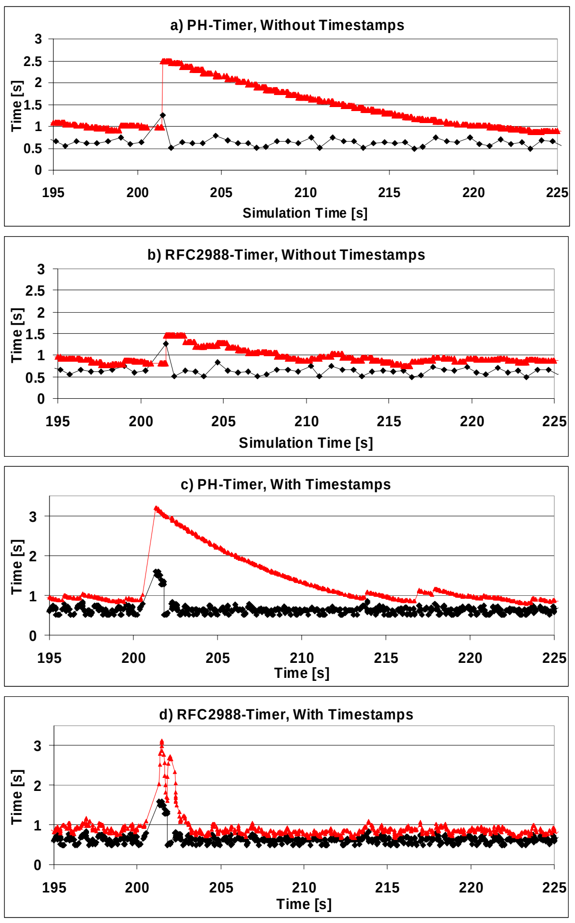


Figure 16. Increase and Decay in Response to a Delay-Spike

comprehensive analysis on the subject.

We then proposed a new algorithm for calculating the RTO, named the Peak-Hopper-RTO, that removes those problems, namely the interaction with timing every packet based on the RTTM mechanism defined in [RFC1323]. This interaction is keeping [RFC1323] and [RFC2988] from advancing on the IETF's standards track. Our results show that in most cases the PH-Timer yields better performance both in terms of quick loss detection when a packet was lost, and in terms of minimizing the number of spurious timeouts. The PH-Timer on average detects a lost packet up to one RTT quicker than the RFC2988-Timer while at the same time causing less or at least not more spurious timeouts. Moreover, the loss detection times for the PH-Timer are much more closely spread around the average whereas the RFC2988-RTO often generates "RTO outliers" that lead to exceptionally long loss detection times.

We further demonstrated the benefit of timing every packet as suggested in [RFC1323]. In most scenarios a considerably better retransmission timer performance is achieved when timing every packet instead of only one packet per RTT.

An important issue that we have not addressed in our work is the timer complexity, and hence, the computational effort it requires. Clearly, the Peak-Hopper-Timer requires more operations than the RFC2988-Timer. But, also those operation can be executed in integer arithmetic. Although, we do not expect computational complexity to be an issue for the Peak-Hopper-Timer, it certainly merits further analysis.

Another issue that deserves attention is the rule specified in [RFC1323] for how the TCP receiver should echo timestamps in duplicate ACKs. The current rule does not allow the TCP sender to derive precise RTT samples because the timestamp echoed in duplicate ACKs is from the last segment that arrived in sequence. During loss recovery when only duplicate ACKs return to the TCP sender this effectively masks any RTT changes that may have occurred in the network. This may become a problem especially for TCP connections running with large congestion windows across a path with considerable loss rates. In this case, a large fraction of the ACKs returning to the sender will be duplicate ACKs. Providing ways to derive precise RTT samples from duplicate ACKs would also allow to dynamically update the retransmission timer that is running for the fast retransmit.

In our future work, we plan to further study a number of items. First, we would like to complement our simulations with large-scale Internet measurements. We also would like to evaluate the Peak-Hopper-Timer for SCTP, since in this work we have only focused on TCP. In that context, we plan to investigate alternatives to Karn's backoff algorithm [KP87], [RFC2988]. We are not convinced that such a rapid backoff (by doubling the RTO) is really required. We also question whether the backoff needs to be extended to the current maximum of 60 seconds. Another study item we plan to evaluate is the benefit of making the 'Fader' variable in the PH-Timer definition adaptive to the fraction of retransmits caused by a spurious timeout. This will require a detection scheme for spurious timeouts such as the Eifel detection scheme [RFC3522].

REFERENCES

[AAAB03] M.Allman, K. Avrachenkov, U. Ayesta, J. Blanton, “Early Retransmit for TCP and SCTP”, Internet Draft, draft-allman-tcp-early-rexmt-01.txt, Work In Progress, June 2003

[AP99] M. Allman, V. Paxon, ”On Estimating End-to-End Network Path Properties”, ACM SIGCOMM 1999

[BLM02] J.W. Byers, M. Luby, M. Mitzenmacher, “A Digital Fountain Approach to Asynchronous Reliable Multicast”, IEEE Journal on Selected Areas in Commuinications, Special Issue on Network Support for Multicast Communications, 2002

[Jac88] V. Jacobson, ”Congestion Avoidance and Control”, ACM SIGCOMM, 1988

[LS00] R. Ludwig, K. Sklower, “The Eifel Retransmission Timer”, ACM Computer Communication Review, Vol. 30, No. 3, July 2000

[NS] S. McCanne, S. Floyd. “The Network Simulator – ns-2”, http://www.isi.edu/nsnam/ns/

[PM01] J. Peisa, M. Meyer, ”Analytical Model for TCP File Transfers over UMTS”, IEEE 3GWireless, pp. 42-47, May 2001

[RFC793] J. Postel, “Transmission Control Protocol”, RFC 793, September 1981

[RFC889] D. L. Mills, “Internet Delay Experiments”, RFC 889, December 1983

[RFC1323] V. Jacobson, R. Braden, D. Borman, “TCP Extensions for High Performance”, RFC 1323, May 1992

[RFC2018] M. Mathis, J. Mahdavi, S. Floyd, A. Romanov, “TCP Selective ACKOptions”, RFC 2018, October 1996

[RFC2581] M. Allman, V. Paxon, W. Stevens, “TCP Congestion Control”, RFC 2581, April 1999

[RFC2582] S. Floyd, T. Henderson, “The NewReno Modification to TCP's Fast Recovery Algorithm”, RFC 2582, April 1999

[RFC2883] S. Floyd, J. Mahdavi, M. Mathis, M. Podolsky, “An Extension to the Selective ACK(SACK) Option for TCP”, RFC 2883, July 2000

[RFC2960] R. Stewart, et al., “Stream Control Transmission Protocol”, RFC 2960, October 2000

[RFC2988] V. Paxon, M. Allman, “Computing TCP's Retransmission Timer”, RFC 2988, November 2000

[RFC3042] M. Allman, H. Balakrishnan, S. Floyd, ”Enhancing TCP’s Loss Recovery Using Limited Transmit”, RFC 3042, January 2001

[RFC3168] K. Ramakrishnan, S. Floyd, D. Black, ”The Addition of Explicit Congestion Notification (ECN) to IP”, RFC 3168, September 2001

[RFC3481] H. Inamura, G. Montenegro, R. Ludwig, A. Gurtov, F. Khafizovet, “TCP over Second (2.5G) and Third (3G) Generation Wireless Networks”, RFC3481, February 2003

[RFC3517] E. Blanton, M. Allman, K. Fall, L. Wang, ”A Conservative Selective Acknowledgment (SACK)-based Loss Recovery Algorithm for TCP”, RFC 3517, April 2003

[RFC3522] R. Ludwig, M. Meyer, “The Eifel Detection Algorithm for TCP”, RFC3522, April 2003

[SK02] P. Sarolahti, A. Kuznetsov, “Congestion Control in Linux TCP”, USENIX 2002

[WS95] G. R. Wright, W. R. Stevens, “TCP/IP Illustrated, Volume 2 (The Implementation)”, Addison Wesley, January 1995

[Zha86] L. Zhang, “Why TCP Timers Don’t Work Well”, ACM SIGCOMM 1986