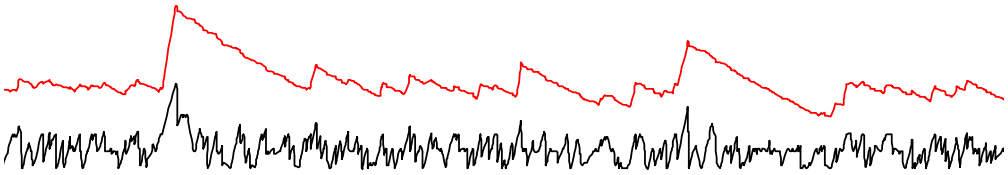
The Peak-Hopper: A New End-to-End Retransmission Timer for Reliable Unicast Transport



Hannes Ekström, Reiner Ludwig
Ericsson Research
Aachen, Germany

***Abstract*— We analyze the RTO algorithm standardized for TCP and SCTP [RFC2988] that is widely deployed in the Internet, referred to as the RFC2988-RTO in this paper. We briefly demonstrate three well-known problems of the RFC2988-RTO. We then develop a new RTO algorithm called the Peak-Hopper-RTO (PH-RTO) that eliminates the mentioned problems. Through extensive simulations in ns2, we evaluate the effectiveness of both RTO algorithms. The key advantage of the PH-RTO over the RFC2988-RTO is its predictability. Although the RFC2988-RTO often exhibits low loss detection times, it also exhibits frequent "RTO outliers" that can lead to exceptionally long loss detection times. The loss detection times of the PH-RTO are much more closely spread around the mean. Finally, our results show that the PH-RTO is much more robust to sudden delay spikes that are particularly common in wireless networks.**

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 timing of every packet, e.g. when using the TCP Timestamps option [RFC1323]. This interaction often causes the RTO to follow the RTT too closely. Other problems of the RFC2988-RTO that exist with and without using timestamps are its sluggish response to delay spikes, and its overly conservative response to sharp drops of the RTT. Addressing these problems has been the motivation for the work presented in this paper.

It is well known that a retransmission timer is only a last resort, and that 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 [RFC2581], NewReno [RFC2582], limited transmit [RFC3042], SACK-based loss recovery [RFC2018][RFC3517], and early retransmit [AAAB03]. For bulk data transfers in low loss environments, these schemes have made TCP's performance largely 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. However, the TCP performance for interactive applications or even bulk data transfers in high loss environments still relies on the retransmission timer. This is because such connections often have too few or no acknowledgements (ACKs) in flight to trigger any advanced loss recovery scheme.

We propose a new algorithm for calculating the RTO, named the Peak-Hopper-RTO (PH-RTO) that removes the mentioned interaction with the timing of every segment, and also solves the other known problems with the current RTO algorithm. In our work we have only focused on the RTO algorithm, and have reused from [RFC2988] the rules how Round-Trip Time (RTT) samples are taken, the timer granularity (G), and 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 the RFC2988- and the PH-Timer in various scenarios. Previous work has shown that the performance of the RFC2988-Timer is dominated by the fixed RTO minimum of 1 second [AP99]. In order to focus on the RTO calculation algorithm itself, we have replaced that minimum with an adaptive value as used in the FreeBSD implementation of TCP. As motivated in Section V.A, we have also modified the initialization of the RFC2988-Timer. 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 and the initialization.

The results show that key advantage of the PH-RTO over the RFC2988-RTO is its predictability. Although the RFC2988-RTO often exhibits low loss detection times, it also exhibits frequent "RTO outliers" that can lead to exceptionally long loss detection times. The loss detection times of the PH-RTO are much more closely spread around the mean. Finally, our results show that the PH-RTO is much more robust to sudden delay spikes that are particularly common in wireless networks.

The paper is organized as follows. In Section II, we review known problems of the RFC2988-RTO. In Section III, we introduce the PH-Timer where we first motivate our design decisions, and then explain its features in detail. In Section IV, 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 V, we present and discuss our simulation results. Finally, in Section VI, 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 one-second minimum defined for the RTO in [RFC2988]. We then discuss a feature of the RFC2988-Timer that results from the rules that govern the management of the retransmission timer. In the last 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:

$RTTVAR \leftarrow 3/4 * RTTVAR + 1/4 * |SRTT - RTT_{SAMPLE}|$

$SRTT \leftarrow 7/8 * SRTT + 1/8 * RTT_{SAMPLE}$

$RTO \leftarrow SRTT + \max(G, 4*RTTVAR)$

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

A. The RTO Minimum

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 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 been changed to a fixed (!) value of 1 second. In our simulations we have replaced it with "(RTT-Sample + 2 * G)" as used in the FreeBSD implementation of TCP.

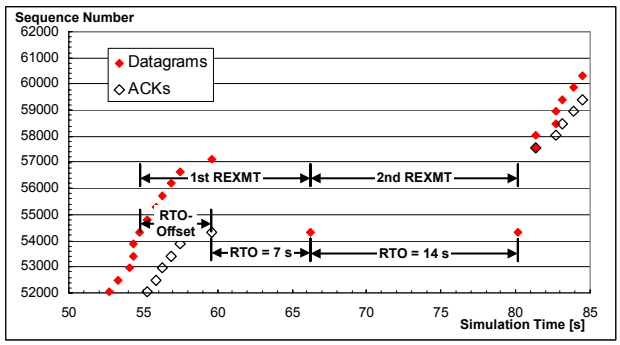


Figure 1. The RTO-Offset

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 should not rely on a fixed minimum.

B. The Implicit RTO-Offset

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 "RTO-Offset" in Figure 1. The retransmission timer expires after a time that is the sum of the RTO-Offset, which actually is "the age of the oldest outstanding segment", and the calculated RTO.

In previous work, we had identified this as a "bug" [LS00], but our analysis in this study 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. It is 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. Note that in all plots in this paper that show a graph of RTO, the RTO-Offset is not depicted, i.e., we always only plot the calculated RTO.

C. Fixed 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. These constants have been chosen under the

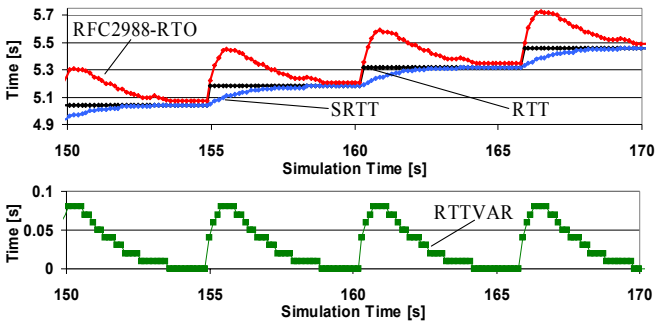


Figure 2. Fixed gains don't scale

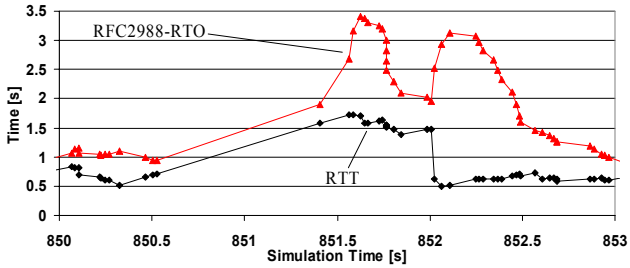


Figure 3. Sluggish response to delay-spikes

assumption that the TCP sender only times one packet per RTT [Jac88]. However, if the sender times every packet, the fixed gains often lead to an RTO that is too low. The problem is aggravated as the congestion window increases, and more RTT samples are taken per RTT. In such cases, 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 this scenario a single TCP connection runs across a bottleneck with a high bandwidth-delay product, and every packet is timed. 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 was shown in our previous work [LS00]. 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.

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 RTO-Offset depicted in Figure 1, this delay spike does not lead to a spurious timeout. Note that despite the rather large delay spike, the resulting RTO computed shortly before second 851.5 lies just above the RTT. 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 has already been observed 20 years ago [RFC889], [Zha86]. This problem is

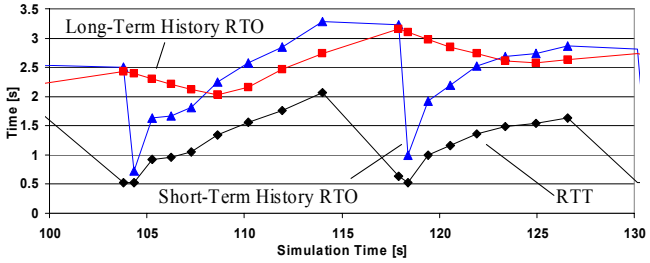


Figure 4. The PH-RTO is calculated as the envelope of the short- and long-term history RTO curves

even more pronounced without the use of timestamps since it takes the RFC2988-RTO longer (in time) to adapt to an appropriate level, if the increased RTT persists.

Note that the problem discussed in the previous sub-section is also visible in the steep decline of the RTO between 851.5 and 852 seconds in Figure 3. 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 is depicted 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 [LS00]. In contrast to the problems explained in the previous two sub-sections, this problem leads to an RTO that is too conservative. This problem is more pronounced with the use of timestamps since the SRTT converges to the RTT more rapidly in this case.

III. THE PEAK-HOPPER

A. The Basic Idea

We have adopted a key idea from previous work for the design of the Peak-Hopper-RTO. In [RFC889] Mills suggests that the RTO should be responsive to upward-going trends in the RTT and less responsive to downward-going trends. The main novelty introduced by the Peak-Hopper is that it essentially runs two RTO algorithms in parallel as shown in Figure 4. 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 second 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, i.e., the PH-RTO follows the envelope of the short-term and long-term history RTO curves. Another novelty of the PH-RTO is that if the short-term history 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 it minimizes packet length effects [RFC889], [Zha86].

B. *The Details - Calculating the PH-RTO*

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

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

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

$$B \leftarrow \min\left(\max(2*\delta, D*B), B_{\max}\right) \qquad \text{(Step 3)}$$

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

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

For each new RTT sample, RTT_{sample}, we calculate the normalized change, δ, from the previous RTT sample, $RTT_{previous}$, as shown in Step 1. We use δ as a measure of the short-term RTT history. The rationale for monitoring this change is that if δ is positive and large enough, i.e., if there has been a large increase in RTT, then the relative distance between RTO and RTT should be increased.

In Step 2, we calculate a decay factor, D, which controls the decay of the RTO. For this purpose, we 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 than 1.

We also introduce a variable S, which is set equal to the number of RTT samples that are expected to be collected from the packets currently in flight. If the timestamp option is not used[1] only one RTT sample is taken per flight, hence S is set to 1. However, if the timestamps are enabled, S is set to the flight size. The flight size corresponds to the number of RTT samples collected, provided no packets are lost, and the delayed ACK option is not enabled. Thus, the decay of the RTO is scaled with the flight size: if the flight is large, each arriving ACK only decays the RTO by a little, since D is large. The opposite is true for an ACK arriving when the flight size is small. We previously proposed this scaling-concept in [LS00]. If the delayed ACK option is enabled at the receiver, often fewer ACKs arrive at the sender, ultimately resulting in a more conservative decay of the RTO. Note that since F is greater than 1 and S is greater or equal to 1, D is always smaller than 1.

In Step 3, we calculate a booster variable B, which determines by how much the RTO should be dominated by the short-term history. We set B to the maximum of $2*\delta$ and the decayed value of the current B. However, we also prevent B from assuming values greater than B_{max}. We set B_{max} to 0.5 if timestamps are enabled and to 1 if timestamps are not enabled. These numbers have been chosen to account for a worst case, where the RTT doubles between two consecutive flights, e.g.

[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.

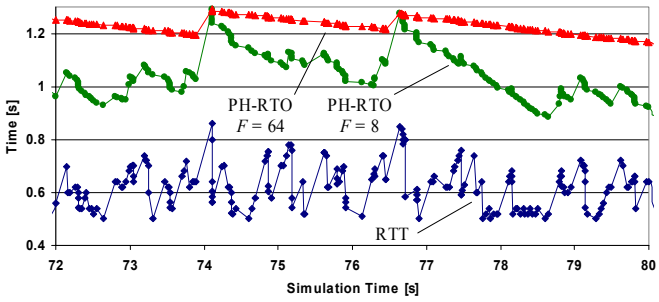


Figure 5. Effect on PH-RTO of different Fader variables

during slow-start. Note that B is always greater than 0. Consequently, a large RTT increase yields a large B, which in turn makes the short-term history dominate the RTO (Step 5).

In Step 4, we set RTT_{max} to the maximum of the new RTT sample and the previous RTT sample. This is done to add some conservativeness to the RTO calculation. RTT_{max} is later used to calculate the short-term history of the RTT.

Step 5 is the most central one to the PH-RTO algorithm since it realizes the basic idea of the PH-Timer as described in the previous sub-section. We set RTO to the maximum of the 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, B determines how high the RTO should hop. In this case, the long-term history is erased and re-initialized with the newly calculated RTO value. This process is depicted in Figure 4.

The rationale for erasing the long-term history at this point is as follows: a large RTT increase is a strong indication that the RTO estimate stored in the long-term history is no longer sufficiently conservative. The logical consequence is that the PH-RTO "hops" upon detection of such an RTT peak. In effect, this means that the PH-RTO hops from one RTT peak to the next and decays in between. This particular characteristic of the PH-RTO algorithm has also provided its name: The Peak-Hopper. In Step 5, we also ensure that the calculated RTO does not fall below the defined minimum, RTO_{min}. We discuss the setting of RTO_{min} in Section III.D.

One of the key benefits of the PH-RTO is the simple and intuitive manner in which it can be configured. In fact, the PH-RTO only maintains a single parameter that needs to be configured: the fader variable F. As illustrated in Figure 5, where setting the F = 64 decays the PH-RTO considerably slower than setting F = 8, F determines how fast the PH-RTO decays. 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 without timestamps, a slightly higher value of F is required. This is motivated by the fact that when only one packet per flight is timed, the RTO calculation has to suffice with less information about the network state. Therefore, we need to make a more conservative RTT estimate,

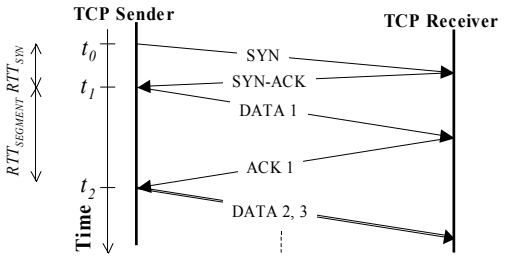


Figure 6. Setup of a TCP Connection

and have hence settled for $F = 24$ when timestamps are not enabled in our simulations.

Another benefit of the PH-timer is that after a spurious timeout, no active adaptation of the RTO is needed as described for the RFC2988-Timer in [LG03]. Such a spurious timeout is commonly caused by a delay-spike. Using the PH-RTO, such a delay spike will have increased the booster variable, B substantially. Hence, the regular PH-RTO calculation yields conservative RTO values following a spurious timeout, thereby eliminating the need for any active adaptation of the RTO.

C. Initializing the RTO

The initialization of the PH-RTO differs from the initialization specified in [RFC2988]. We changed the initialization in order to minimize the impact of the packet length effect [RFC889][Zha86][RFC3481], which may lead to an inaccurate RTO calculation on low bandwidth paths. In Figure 6 we illustrate the initialization process and the terms we use in the following description.

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, the RTT measurement of the first data segment often is considerably larger than that of the relatively small SYN segment. Hence, for such paths we can expect $RTT_{SEGMENT} > RTT_{SYN}$, where $RTT_{SEGMENT}$ is the first RTT measurement of a data segment. This motivates the conservative setting of the RTO for the first data segment.

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

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

where B_{INIT} is the initial value of the Booster and $RTT_{SEGMENT}$ is the RTT measured for the first data segment. Especially for bandwidth-dominated paths, $RTT_{SEGMENT}$ serves as a better basis upon which to make predictions on future RTT samples. From this point on, the RTO is calculated as described in the previous sub-section.

In our simulations, we have set B_{INIT} statically to 0.25. However, we envision that a TCP sender could maintain a time-varying B_{INIT} in order to dynamically strike a tradeoff between spurious timeouts in the initial phase of the connection and promptness of the loss detection. We leave this idea for further study.

D. The RTO Minimum

We define the minimum RTO, RTO_{min}, as

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

where RTT_{max} is the maximum of the two last RTT measurements and G is the granularity of the timer 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 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]. 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]. In this section, the metrics and the simulation scenarios are described.

A. TCP Configuration in Simulations

The TCP senders used in the simulations all make use of the following options or enhancements:

- Basic congestion control according to [RFC2581]
- Limited transmit [RFC3042]
- SACK error recovery as defined in [RFC3517]

We have not enabled the delayed ACK option in the simulations. As already discussed, the PH-RTO is slightly less conservative, i.e. it decays faster, without delayed ACK.

In the analysis, we are also interested in evaluating the effect of using the timestamps option on the retransmission timer performance. Hence, we simulate both with and without this option enabled. In the simulations, we use a heartbeat timer with a granularity, G, of 10ms. In [AP99] and [Jac01], this setting has been shown to give good performance.

B. Simulation Scenarios

We evaluate the performance of the retransmission timers in scenarios that exhibit a wide range of different RTT patterns. We have defined three scenarios, as described below.

1) Scenario A – High Degree of Statistical Multiplexing

In this scenario, N flows share a bottleneck link, as depicted in Figure 7. Each of the N senders (S_i, $i \in \{1 \ldots N\}$), communicates with one receiver (R_i, $i \in \{1 \ldots N\}$). All links

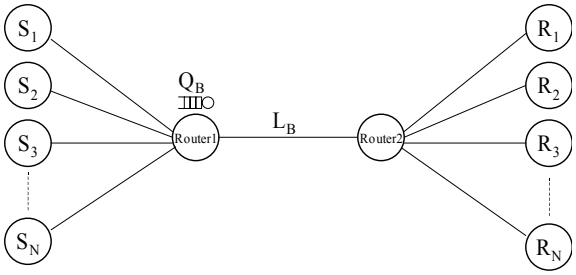


Figure 7. Simulation topology used in Scenario A

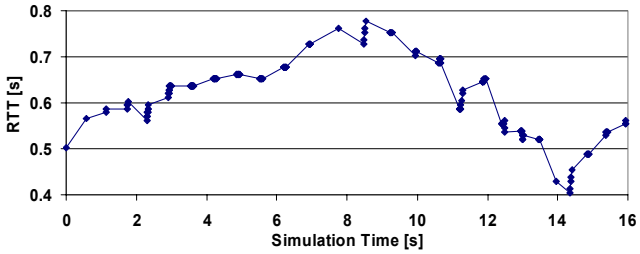


Figure 8. Typical RTT-Pattern in Scenario A

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. In the simulations, *N* is varied between 940 to 1036 in order to obtain results for different network loads. The bottleneck link queue, Q_B, is a RED queue with *thresh*, *maxthresh_*, *q_weight_* and *linterm_* set to 30packets, 100packets, 0.002 and 10 respectively. Furthermore, the "*gentle_*" mode of RED available in ns2 was used. The queue applies a drop-from-front policy.

In a single simulation run *N*/2 senders use the RFC2988-Timer, of which half use a bulk workload model and half use an interactive workload model to generate traffic. The other *N*/2 senders use the PH-Timer with the same traffic distribution.

In the bulk workload model, each sender transmits an FTP file consisting of exactly 200 IP packets to its receiver. A typical RTT pattern arising from this scenario is shown in Figure 8. In the interactive workload model, each sender transmits 40 IP packets to its receiver in a ping-pong fashion, i.e. the sender waits for the ACK of a specific packet before transmitting the next packet. Each sender starts its data transmission at a time that is uniformly distributed in the interval (0, 1000) seconds.

2) *Scenario B – WCDMA Link*

In this scenario, we simulate a single sender, S_1, which communicates with a single receiver, R_1 as shown in Figure 9. The link-layer of L_B in Figure 9 is a model of a WCDMA link implemented in the ns-2 simulator. The WCDMA-link model is based on the results in [PM01]. When calculating the transmission delay of an IP-packet, this model takes into account delays caused by link layer block errors and the ensuing link layer retransmissions. Two nominal bit-rates are simulated: 64kbps and 384kbps. Q_B is a passive queue with a size of approximately one pipe-capacity. The TCP sender starts

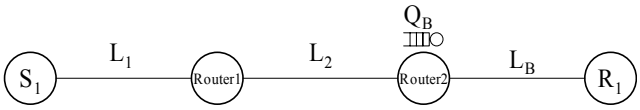


Figure 9. Simulation topology used in Scenarios B

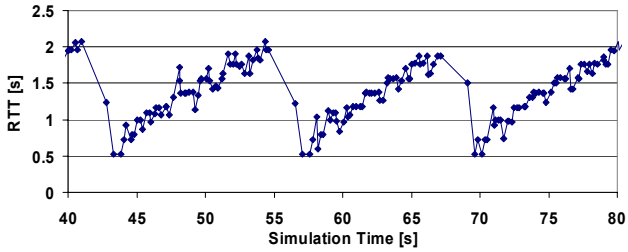


Figure 10. Typical RTT-Pattern in Scenario B

sending data at t=0 seconds and stops at t=400 seconds. During this time, it is never application limited.

The resulting RTT seen by the TCP sender is determined by the queuing delay at Q_B, and additionally by the variable transmission delay arising from the modeled link layer retransmissions. A typical RTT pattern for this scenario is shown in Figure 10.

3) *Scenario C – Delay Spikes*

The bottleneck link, L_B, used in this scenario is the same as in Scenario B, i.e., it models a WCDMA-link. However, the capacity of the bottleneck link queue, Q_B, is larger than the advertised window of the receiver R_1. Hence, no packets are dropped at Q_B. The main difference in this scenario to the previous one is that the bottleneck link L_B is interrupted on the data path (from Router2 to R_1) at regular intervals. Such transient link outages may often occur in wireless networks [RFC3481]. The obvious effect of such link outages is that the TCP sender observes delay-spikes. The bottleneck link is interrupted with a periodicity P_B and every outage has a duration *d*, where P_B and *d* are constant for each simulation run. In the simulations we set, P_B to 50 seconds and vary *d* between 0.5 and 3 seconds. The links L_1 and L_2 both have a latency of 50 ms. We simulate two different rates at the bottleneck link, L_B: 64 and 384 kbps. The topology for this scenario is shown in Figure 11 and a typical RTT-pattern for this scenario is shown in Figure 12.

C. *Performance Metrics*

We assess the performance of the retransmission timer according to the following metrics:

- Accuracy
- Promptness
- Predictability
- Proximity

We will discuss these metrics in the following sub-sections. Note that when calculating these metrics, we only consider timeouts that initiate an error recovery phase, i.e., we neither

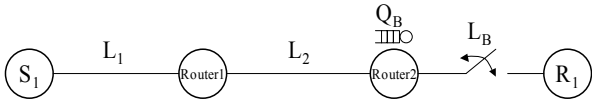


Figure 11. Simulation topology used in Scenario C

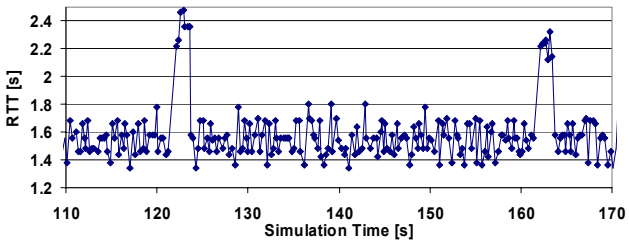


Figure 12. Typical RTT-Pattern in Scenario C

consider multiple timeouts for the same segment nor timeouts that occur during fast recovery.

1) *Assessing the Accuracy*

With accuracy we refer to the retransmission timers ability to avoid spurious timeouts. In order to assess the accuracy, we measure the normalized 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 by all flows during a simulation, and *#DataSegments* is the total number of data segments injected into the network by all senders during a simulation. We express B_{norm} in terms of percent.

2) *Assessing the Promptness*

With promptness we refer to the retransmission timers ability to provide timely loss detection. We call a timeout ”necessary” if it is not spurious according to the definition of a spurious timeout provided in [RFC3522]. In order to assess the promptness of the retransmission timer, i.e., the timeliness of its response, we define the normalized loss detection 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 last (re-)started, and RTT_i is the last RTT sample taken before the timeout. We calculate the mean normalized loss detection time for a necessary timeout as

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

where 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 since it was last started until it expired. The metric W is expressed in multiples of the last RTT measurement prior to the timeout. This metric differs slightly to the one used in [AP99], since we do not include in W_i the RTO-Offset (see Section II.B). Note that we exclude timeouts that occur on the initial SYN segment from this analysis, since when such a timeout occurs, the first RTT measurement has not yet been made, hence W_i is undefined.

3) *Assessing the Predictability*

With predictability we refer to the retransmission timers ability to minimize the spread of the loss detection times around its average. In order to assess the predictability we define $W_{95\%}$ as the value for which 95 percent of all W_i values collected in a simulation run are smaller than $W_{95\%}$. A high $W_{95\%}$ indicates 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 the mean loss detection time.

4) *Assessing the Proximity*

With proximity we refer to the retransmission timers ability to minimize the relative distance between the RTT and RTO. For simulation runs where no or only a few timeout events are observed, we cannot fairly judge the performance of the retransmission timers based on the accuracy, promptness and predictability metrics. Hence, we need another metric to judge the performance of the retransmission timer.

In such cases, we choose to use the proximity of the RTO to the RTT as a performance metric. For the *j*:th RTO calculation in a simulation run, we define P_j as

$$P_j = \frac{RTO_j}{RTT_j}.$$

RTO_j is the value resulting from the *j*:th RTO calculation and RTT_j is the last RTT sample prior to calculating RTO_j. We then calculate the proximity, P, as

$$P = \frac{1}{N}\sum_{j=1}^{N} P_j,$$

where N is the number of times that the RTO is calculated in the simulation run.

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 on average 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 calculated), while W_i is only sampled when a necessary timeout occurs.

V. RESULTS

In this section, we compare the performance of the RFC2988-Timer to that of the PH-Timer. As is shown in sub-section V.A, the performance of the RFC2988-Timer is impeded by its conservative initialization. For this reason, in all the evaluations in the following section, we have modified the initialization of the RFC2988-Timer as described in Section V.A. This allows us the focus the evaluation on the RTO-calculation itself, eliminating artifacts arising from a too conservative initialization. In sub-sections V.B-V.D we present

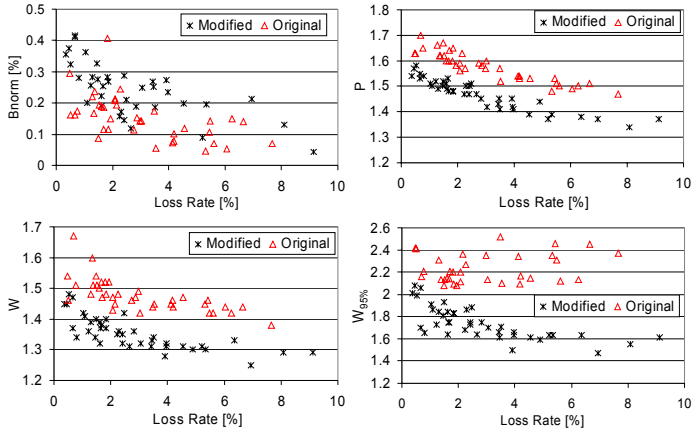


Figure 13. Results for interactive workload users with timestamps. The plots show the results for users with the original RTO initialization from [RFC2988] and for users with the modified RTO initialization.

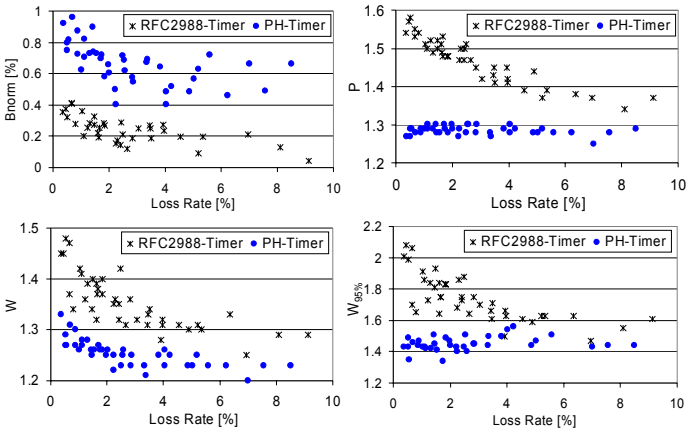


Figure 14. Results for interactive workload users with timestamps

the simulation results from the three different simulation scenarios.

A. Modifying the RFC2988 Initialization

We are primarily interested in comparing the effectiveness of the PH-RTO with that of the RFC2988-RTO. However, we found that the performance of the RFC2988-RTO is greatly impeded by its conservative initialization as defined in [RFC2988]. This is especially evident for short-lived flows, as produced by the interactive workload model in simulation Scenario A. With this initialization of the RFC2988-RTO, the only conclusion that we could draw is that "in the simulated scenarios, the initialization of the RFC2988-RTO makes the retransmission timer very conservative". In the simulations, we therefore decided to modify the initialization of the RFC2988-RTO in order to remove the artifacts arising from its conservative initialization.

In the modified RFC2988-RTO initialization, we follow the algorithm provided in [RFC2988] until the first RTT measurement from a data segment arrives. We then reinitialize the RTO as follows:

$SRTT \leftarrow RTT_{SEGMENT}$

$RTTVAR \leftarrow RTT_{SEGMENT}/16$

$RTO \leftarrow SRTT + \max(G, 4*RTTVAR)$,

where $RTT_{SEGMENT}$ is the RTT measured for the first data segment. We chose this particular initialization, since this results in an RTO initialization that is equally conservative to the PH-RTO initialization described in Section III.C. With this modification, both RTO algorithms (PH-RTO and RFC2988-RTO) reinitialize the RTO to $1.25*RTT_{SEGMENT}$ when the RTT measurement from the first data segment, $RTT_{SEGMENT}$, has been made. In this way, we have eliminated any artifacts arising from differing initializations and we can therefore compare the two RTO algorithms on common grounds.

The scatter-plots in Figure 13 show that with the original initialization, the RFC2988-RTO performs slightly better in terms of accuracy (top-left B_{norm}-plot). This is not surprising, considering the conservative initialization. However, in terms of proximity, P, promptness, W, and predictability $W_{95\%}$, the RFC2988-RTO performs considerably worse with the original initialization than with the modified initialization. In all following simulation results presented in this paper, we have used the modified initialization for the RFC2988-RTO as outlined in this sub-section and we have replaced the minimum RTO value as explained in Section II.A.

B. Scenario A – High Degree of Statistical Multiplexing

1) Results With Timestamps

From the results presented in Figure 14, it is evident that for the interactive workload model, different tradeoffs have been struck between accuracy and promptness for the two retransmission timers. Each dot in the plot depicts the result from one simulation run. The top left plot shows that the RFC2988-Timer is clearly more accurate than the PH-Timer, generating considerably less spurious retransmissions. We note that for interactive traffic, where only one packet is outstanding at any given time, the only adverse effect of a spurious timeout is the transmission of one unnecessary segment (since only one segment is outstanding, there is no go-back-N). However, the amount of unnecessary traffic put on the network is less than 1% of the total traffic in all simulations. We believe that this is small enough to be neglected.

On the other hand, the PH-Timer clearly outperforms the RFC2988-Timer in terms of P, W and $W_{95\%}$, providing clearly superior proximity, promptness and predictability. We believe that especially for interactive traffic, these are important characteristics of a retransmission timer. In our eyes, the benefits of these short and predictable loss detection times outweigh the penalty paid in terms of generating unnecessary traffic.

The results for senders using the bulk workload model did not differ significantly between the two retransmission timers in any of the four metrics. For this reason, we do not show the corresponding plots. However, it is noteworthy that up to 90% of all spurious timeouts that occurred in the simulations are "fast timeouts", i.e., a timeout that is followed by the DUPACK that would have triggered the fast retransmit. We

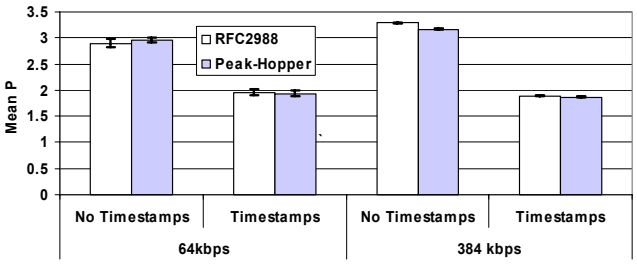


Figure 15. Mean timer proximity values with 95% confidence level

found that the fraction of fast timeouts is especially high when the limited transmit algorithm is enabled. A potential solution to this problem is restarting the retransmission timer on the reception of DUPACKs as described in [Lud02]. We plan to address this issue in future work. Note, however, that using the interactive workload model, no fast timeouts occurred, since at any given time only one packet is outstanding.

2) *Results Without Timestamps*

The results without timestamps do not differ greatly from those already discussed in the previous sub-section with timestamps. For the interactive workload model, the difference in the tradeoff between accuracy and promptness is less pronounced. This was expected, since we have configured the PH-RTO a bit more conservatively when timestamps are not used. For the bulk workload model, there is again no significant performance difference between the two retransmission timers.

C. *Scenario B – WCMDA link*

In this scenario, neither of the retransmission timers produced any spurious timeouts. Apparently, the RTT variations produced by the WCMDA model are not severe enough for this to happen. We therefore base our evaluation of the retransmission timers on the collected timer proximity values, P. Figure 15 shows the mean of the collected P values for the different simulations. We see that there is little difference in this particular metric between the two retransmission timers, i.e. had a timeout been necessary, they would have reacted equally promt (on average).

However, a closer look at the distribution of the P values reveals a significant difference between the two timers. Although the two timers have approximately equal mean P values, the distribution of the P-values of the RFC2988-Timers is considerably more skewed than that of the PH-Timer as can be seen in Figure 16. The skewed nature of the distribution is especially pronounced when timestamps are used. From the figure it is evident that the P-value at the 95th percentile when timestamps are used is 4 for the PH-RTO but 6.5 for the RFC2988-RTO. The corresponding values when no timestamps are used are approximately 7.75 and 8.75. Considering that the RTT in wireless systems may be in the order of a second (including queuing delay), this means that the RFC2988-Timer in some cases provides an extremely slow loss recovery. The reason for this skewed distribution is the prediction flaw. As already predicted in Section II.E, the effect of this flaw is more pronounced when timestamps are used.

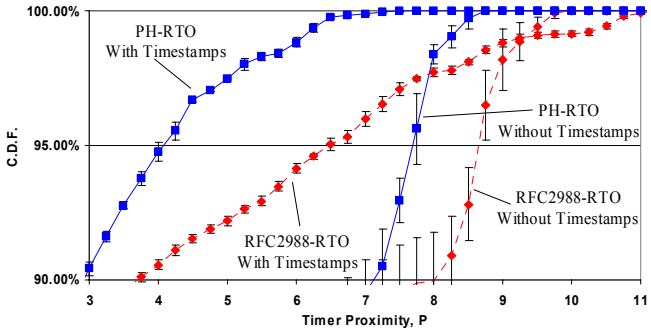


Figure 16. Cumulative distribution function at 95% confidence level for P-values

We note that the performance of both retransmission timers is improved when using timestamps, i.e. the mean RTO-proximity is decreased by up to 40%.

D. *Scenario C – Deterministic Delay Spikes*

In this scenario, we are interested in evaluating the robustness of the timers to delay spikes. As a performance metric, we calculate the percentage of link outage events that generate a spurious timeout. Since this metric is specific to this scenario, it was not introduced in Section IV. Remember that in this experiment we produce a link outage periodically every 50 seconds. With such a high periodicity, both timers "forget" the previous delay spike before encountering the next one. In Figure 17 a) and Figure 17 b), this percentage is plotted as a function of the duration of link outage. In both plots, the two curves to the left and the two curves to the right depict the results for a 384kbps and 64kbps radio bearer, respectively.

For both link rates and independent of whether timestamps are used, the PH-Timer is more robust to delay-spikes than the RFC2988-Timer. For example, as highlighted in Figure 17 a) the PH-Timer is up to 70% less likely to cause a spurious

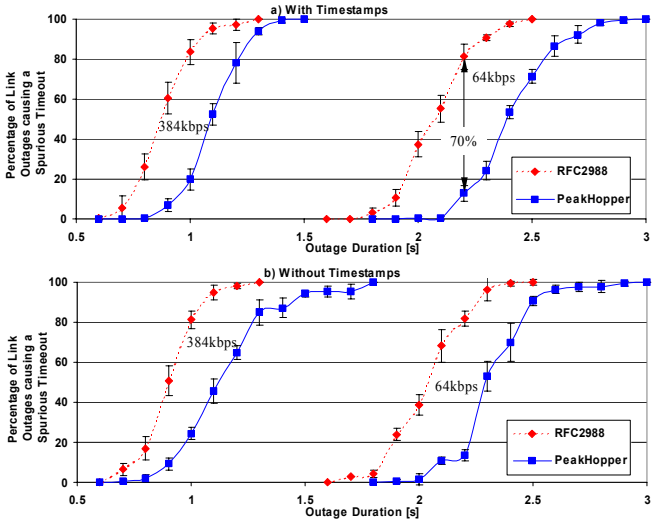


Figure 17. Percentage of link outages that caused a spurious timeout as a function of the outage duration (95% confidence level). a) With timestamps b) Without timestamps

timeout than the RFC2988-Timer with a constant link outage time of 2.2 seconds. The reason for this is that the PH-RTO is more conservative in this scenario.

In order to give a microscopic view of the timer behavior we have made another experiment. The result is depicted in Figure 18. In this experiment, we produce a link outage of duration 0.9 seconds at 20 seconds and 27 seconds, causing two delay-spikes separated by 7 seconds. All four plots depict the RTT (lower curve), and the RTO (upper curve). By visual inspection of each plot 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 second delay-spike to half of that maximum value.

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 (2.7 seconds with timestamps and 3 seconds without timestamps), while the half-life of the RTO without timestamps (10 seconds in Figure 18 a) is comparable to that with timestamps (6.9 seconds in Figure 18 c). In fact, the more drastic increase and conservative decay of the PH-Timer without timestamps are both the result of conscious design choices. Without timestamps we placed an upper bound of the "booster" of 1 compared to 0.5 with timestamps, resulting in a more drastic increase in response to the delay spike. Furthermore without timestamps we have chosen a fader variable value of 24, as opposed to 16 when using timestamps, resulting in a more conservative decay following the increase.

The characteristics of the RFC2988-RTO, on the other hand, change dramatically when using timestamps. First of all, the maximum RTO value ensuing from the delay-spike differs greatly: it is approximately 3.3 seconds with timestamps (Figure 18 d) while the maximum stays at a mere 1.9 seconds without timestamps (Figure 18 b). The second difference is the half-life: the half-life without timestamps is long (7.2 seconds), when compared to the extremely short half-life time when using timestamps (1.8 seconds). The reason for this rapid decay is the scaling flaw described in Section II.

In fact, the short half-life and the less conservative nature of the RFC2988-Timer when using timestamps makes it produce two spurious timeouts in the trace show in Figure 18 d), one for each of the two delay-spikes. This result strengthens our view that with timestamps, the RFC2988-Timer "forgets too fast". Note that in the figures (e.g. Figure 18 c), at the time of the link outage, the RTO has a value far lower than the first RTT measurement obtained from the first "delayed" segment. In these cases, the RTO-Offset "saves" the retransmission timer from expiring prematurely.

The analysis presented above shows that the PH-Timer is able to eliminate the interactions with the timing of every packet. The performance and behavior of the PH-RTO is largely invariant to whether every packet or only one packet per flight is timed. However, as can also be verified in the plots, the higher sampling frequency when using timestamps allows the PH-RTO to set its RTO less conservatively.

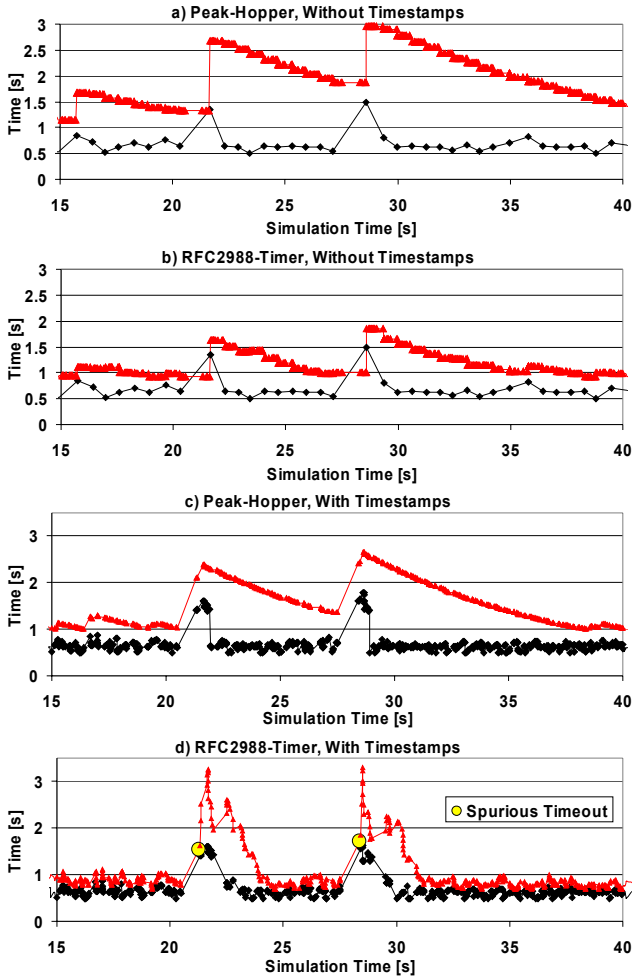


Figure 18. RTO increase and decay in response to delay-spikes

VI. CONCLUSIONS

We have analyzed the RTO algorithm standardized for TCP and SCTP [RFC2988] that is widely deployed in the Internet. Throughout this paper have referred to that algorithm as the RFC2988-RTO. First, we briefly demonstrated three well-known problems of the RFC2988-RTO: (1) its poor interaction with the timing of every segment caused by the use of fixed gains, (2) its sluggish response to delay spikes, and (3) the prediction flaw in response to sharp drops of the RTT. We then developed a new RTO algorithm called the Peak-Hopper-RTO (PH-RTO) that has eliminated the mentioned problems. One explicit design goal for the PH-RTO was to provide a simple way to adjust its conservativeness. We achieved this with a single parameter that we called the "fader": the lower the fader the less conservative the PH-RTO. It might be possible to make the PH-RTO more effective for long-lived flows by making the fader adaptive. However, in our simulations we have only used two different fixed values for the fader: one if the Timestamps option [RFC1323] was enabled and thus every segment was timed, and a larger one if only one segment per RTT was timed.

Through extensive simulations in ns2 that covered a number of common scenarios, we have evaluated the effectiveness of both RTO algorithms in terms of

- accuracy (fraction of spurious timeouts)
- promptness (mean loss detection time)
- predictability (spread of the loss detection times around the mean), and
- proximity (mean ratio of the RTO to the RTT).

The latter metric is useful in scenarios with low packet loss rates, where the metrics promptness and predictability are less meaningful. To have a fair comparison we have replaced the RTO minimum of 1 second that is required for the RFC2988-RTO with a similar minimum as used for the PH-RTO. We then found that the effectiveness of the RFC2988-RTO is impeded by its conservative initialization. To also remove this bias, we have changed the initialization of the RFC2988-RTO to make it equally conservative to the initialization of the PH-RTO.

The key advantage of the PH-RTO over the RFC2988-RTO is its predictability. Although, the RFC2988-RTO often exhibits low loss detection times, it also exhibits frequent "RTO outliers" that can lead to exceptionally long loss detection times. The loss detection times of the PH-RTO are much more closely spread around the mean. In one scenario with request-/response-style traffic 95 percent of all packet losses were detected in less than 1.5 RTT for the PH-RTO compared to up to 2.1 RTTs for the RFC2988-RTO. The "unpredictability" of the RFC2988-RTO can also be seen when evaluating the proximity. In one simulation scenario where the bottleneck was a wide-area wireless access link the 95th percentile of the proximity for the RFC2988-RTO was 6.5 RTTs compared to 4 RTTs for the PH-RTO. This is a significant difference since in wide-area wireless networks TCP's RTT (including queueing delays) is often on the order of a second. The difference should be noticeable to users of interactive applications running across such networks when the request or the response that often fits into a single packet is lost. Finally, our results show that the PH-RTO is much more robust to sudden delay spikes that are particularly common in wireless networks.

Although we have not explicitly showed this in the results presented in this paper, we have found in our simulations that the timing of every packet as suggested in [RFC1323] improves the effectiveness of the PH-RTO. The use of timestamps has almost always resulted in shorter loss detection times, and as shown in Section V.C, in scenarios with low packet loss rates the proximity is often greatly reduced. Another related observation we made in our simulations is that enabling the Limited Transmit algorithm [RFC3042] can lead to a very large number of fast timeouts [Lud02]. A "fast timeout" is a timeout event after which the TCP sender receives the duplicate ACK that would have triggered a fast retransmit. In some of our simulation runs more than 90 percent if all timeouts where fast timeouts. In our future work we will study this in more detail, e.g., we will study whether the retransmission timer should be restarted when the Limited Transmit algorithm sends a new data segment in response to the first and second duplicate ACK as suggested in [GL02].

An important issue that we have not addressed in our work is the complexity of the PH-RTO, and hence, the computational effort it requires. Clearly, the PH-RTO requires more operations than the RFC2988-RTO. But, also those operation can be executed in integer arithmetic. Although we do not expect computational complexity to be an issue for the PH-RTO, 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. 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 study a number of related issues. First, we have planned to complement the simulations presented in this paper with large-scale Internet measurements. Second, we have already started a simulation-based evaluation of the PH-RTO for SCTP. We expect that especially the predictability of the PH-RTO will benefit the SCTP failover performance. In that context, we also plan to investigate a less conservative alternative to the exponential backoff algorithm [Jac88], [RFC2988]. We are not convinced that such a rapid backoff (doubling the RTO) is really required. We also question whether the backoff needs to be extended to the current maximum of 60 seconds. Another issue that we have started to study is the idea of adding redundancy to TCP in the form of "potentially spurious retransmissions" [Lud03]. In this context, we have developed a scheme called the Quick-RTO that allows a TCP sender to reduce the value of the RTO when it is application limited. We currently evaluate the tradeoff between spurious traffic generated by the Quick-RTO scheme, and performance gains for interactive applications.

ACKNOWLEDGEMENTS

We thank Andrei Gurtov for many fruitful discussions that have contributed to this work.

REFERENCES

[AAAB03] M.Allman, K. Avrachenkov, U. Ayesta, J. Blanton, "Early Retransmit for TCP and SCTP", 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 Communications, Special Issue on Network Support for Multicast Communications, 2002

[GL02] A. Gurtov, R. Ludwig, “Making TCP Robust Against Delay Spikes”, draft-gurtov-tsvwg-tcp-delay-spikes-00.txt, work in progress, February 2002.

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

[Jac01] S. Jacobsson, ”Performance Evaluation of the Eifel Retransmission Timer”, Master’s Thesis, Aachen University of Technology, October 2001

[LG03] R. Ludwig, A. Gurtov, ”The Eifel Response Algorithm for TCP”, Internet Draft, Work in Progress.

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

[Lud02] R. Ludwig, ”Responding to Fast Timeouts in TCP”, draft-ludwig-tsvwg-tcp-fast-timeouts-00.txt, work in progress, July 2002

[Lud03] R. Ludwig, ”Adding Redundancy to TCP & SCTP?”, Talk at the 56. IETF Meeting (TSVWG), March 2003.

[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

[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

[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