Available online at www.sciencedirect.com
ScienceDirect
Computer Networks 51 (2007) 2033–2048
Computer Networks
www.elsevier.com/locate/comnet

Why TCP timers (still) don't work well

Ioannis Psaras *, Vassilis Tsaoussidis

Demokritos University of Thrace, Department of Electrical and Computer Engineering, 2 Vas. Sofias Str., 67100 Xanthi, Greece

Received 15 April 2006; received in revised form 29 September 2006; accepted 5 October 2006
Available online 13 November 2006

Responsible Editor: R. Sivakumar

Abstract

We argue that the design principles of the TCP timeout algorithm are based solely on RTT estimations and may lead to flow synchronization, unnecessary retransmission effort and unfair resource allocation. We present a new Window-Based Retransmission Timeout algorithm (WB-RTO) for TCP, which exhibits two major properties: (i) it cancels retransmission synchronization, which dominates when resource demand exceeds resource supply and (ii) it reschedules flows on the basis of their contribution to congestion. WB-RTO achieves better fairness and slightly better goodput with significant less retransmission effort.

Keywords: TCP; Congestion control; Retransmission timeout; Flow contention; Packet scheduling

1. Introduction

The retransmission timeout policy of standard-TCP [1] is governed by the rules of RFC 2988 [2]. The algorithm is based solely on RTT measurements, trying to capture dynamic network conditions by measuring the variation of the RTT samples. In particular, the Retransmission Timeout is calculated upon each ACK arrival after smoothing out the measured samples, and weighting the recent history. This way the timeout reflects the weighted average delay currently in the network, rather than the instantaneous network delay [3]. Furthermore, the timeout takes into account the RTT variation measured during the previous transmission rounds. More precisely, upon each ACK arrival, the sender:

- updates the RTT Variation

$$\mathrm{RTTVAR} = 3/4 \times \mathrm{RTTVAR} + 1/4 \times |\mathrm{SRTT} - \mathrm{RTT}_{\mathrm{SAMPLE}}|, \quad (1)$$

- smooths the expected RTT, prior to calculating the timeout

$$\mathrm{SRTT} = 7/8 \times \mathrm{SRTT} + 1/8 \times \mathrm{RTT}_{\mathrm{SAMPLE}} \quad (2)$$

- and finally, calculates the Retransmission Timeout value

$$\mathrm{RTO} = \mathrm{SRTT} + \max(G, 4 \times \mathrm{RTTVAR}), \quad (3)$$

* Corresponding author. Tel.: +30 6942 473072.
E-mail addresses: ipsaras@ee.duth.gr (I. Psaras), vtsaousi@ee.duth.gr (V. Tsaoussidis).

1389-1286/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2006.10.006

where, RTTVAR holds the RTT variation and SRTT the smoothed RTT. In Eq. (3), G denotes the timer granularity, which is incorporated in order to assign a lower bound for the retransmission timeout to protect it against spurious expirations (i.e., when RTT equals the timer granularity) [4].

Although the design of the timeout algorithm has been studied extensively in the past (e.g [5–7,3,8,2,9,10]), the association with its inherent scheduling properties has not really been evaluated adequately. Instead, much attention has been paid on its ability to reflect network delay accurately (e.g. [11–16]), allowing for speedy retransmission when conditions permit and avoiding double submission due to early expiration. However, network delay as it is captured by measuring the RTT alone, cannot always reflect the level of network contention [17,18]. We note that there is no strict bi-directional association between *network contention* and *network delay*. That is, high contention may lead to large delay, but large delay is not always the result of high contention. Consider, for example, a high-contended link where many flows compete for network resources. Timeout events will happen frequently and many flows may experience timeout expirations simultaneously. Hence, immediately after retransmission, contention may prevail again. Alternatively, congestion caused by the increasing rates of just two flows will result in lower demand after timeout expiration. Thus, the level of contention, after retransmission, is high when contention is high. A timeout that incorporates the level of contention has the potential to administer that link better.

The problem of scheduling as it is associated with timeout has another dimension as well. When flows enter synchronously, or become synchronized due to a congestion event, the timeout is adjusted accordingly for all participating flows. Since the buffers, during congestion, are more or less occupied, queuing delay approaches similar values for most packets. Thus, the sample RTT measurements may leave little space for timeout differentiation among the participating flows, leading to possibly synchronized retransmissions. Therefore, fairness cannot be guaranteed: flows are not randomly scheduled, but instead, are possibly partitioned into two groups: the one consisted of a number of flows that suffice to exploit available resources (link, buffer) and the other consisted of the remaining flows that continuously attempt to transmit unsuccessfully. As a result, flows that enter a system simultaneously will be ordered in the queue and possibly follow the same order throughout the upcoming transmission rounds. Furthermore, flows that enter the system when the buffer is fully utilized, may also be excluded in the next rounds as well, for the same reason. Current timeout scheduling may become very deterministic, allowing only a particular set of participating flows to utilize the link.

We evaluate this hypothesis experimentally. Our scenario involves five high-demanding sessions over a low *Delay* × *Bandwidth* product link of 10 packets. Buffers are set in accordance, to hold up to 10 packets. Due to limited resource supply, the demand should be adjusted to an average of 2 packets per window per flow. In Fig. 1, we monitor the sequence numbers to capture the progress in time. We highlight the interval between 500 and 515 s, out of a simulation run that lasts 1500 s. We observed, though, that the sequence number progress is similar throughout the whole simulation. TCP-RTO scheduling policy results in multiple simultaneous transmissions/retransmissions (see for example, circled packets in Fig. 1). Fig. 2 depicts the measured RTT and the associated timeout adjustments, within the same sample timeslot.

In the current scenario all flows experience the same propagation delay (Fig. 4(a)); due to high level of flow contention the bottleneck buffer (i.e., Router 1) is always full, leading to the same, maximum queuing delay for all flows, as well. Hence, we observed similar RTTs for all flows (Fig. 2(a)) and consequently identical RTOs (Fig. 2(b)), which resulted in flow synchronization. We also observed the impact of propagation delay variations. We have experimentally evaluated groups of flows, where each group corresponds to propagation delay paths, which differ from the propagation delay paths of other groups. However, since they all

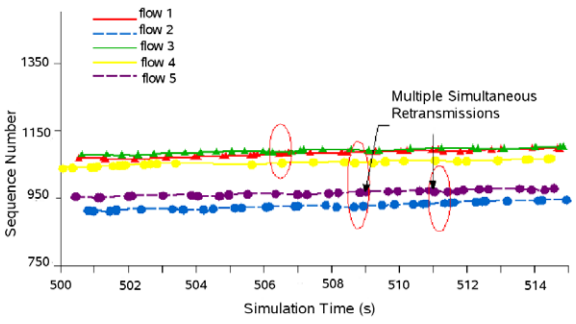

Fig. 1. Sequence number progress.

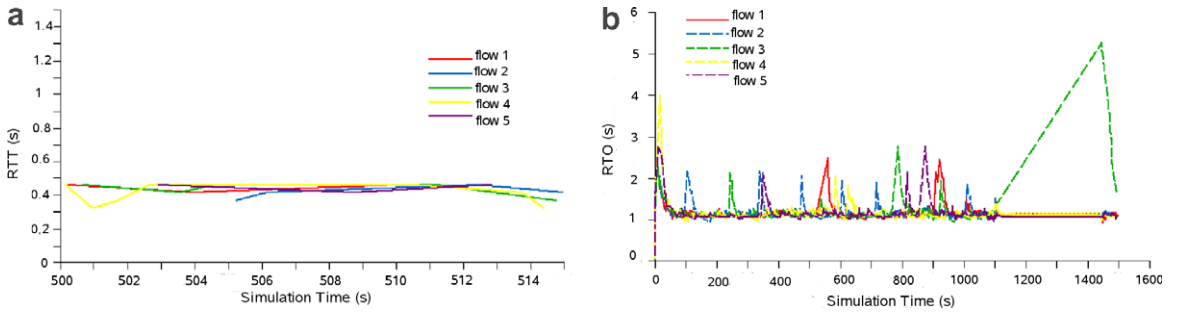

Fig. 2. Reno behavior: (a) Round trip time (in seconds), (b) Retransmission timeout (in seconds).

transmit through the same bottleneck buffer, we noticed that RTOs tend to equalize for flows within the same group.

Although we do not present it here, we observed the sequence number progress for the entire duration of the experiment: for the last 300 s, the third flow did not advance its sequence number, resulting in unfair system behavior. This is further explained by Fig. 2(b), where we see that flow 3 makes wrong RTO estimation, which results in an extraordinary long timeout wait, during the last 300 s of the experiment. We present the Goodput performance and the retransmission effort of TCP-Reno in Table 1. We observe that: (i) the Goodput performance of TCP is 10 packets per second and (ii) TCP triggers retransmissions twice per second, resulting in 16.6% retransmission overhead.

We argue that flow synchronization is responsible for the system response; we seek to address the retransmission scheduling properties of the transport protocol. Our results show that there exists a lot of space for improvement on the scheduling properties of the TCP-RTO, which we exploit through the current proposal.

We have studied further details and reported problems on the efficiency of the TCP retransmission policy in [17]. In the current paper, we extend our study with a new algorithm, to fix the reported problems. We call our algorithm Window-Based Retransmission Timeout (WB-RTO), due to the fact that its calculation mainly depends on the transmission window. Using this information, the algorithm captures how contention evolves with time and makes a judgment on the contribution of the flow to congestion, presently. Furthermore, the algorithm schedules retransmissions in a randomized manner, to avoid link capture effects. The aforementioned mechanisms are coupled with RTT measurements in order to guarantee a lower bound against spurious timeouts. Our results demonstrate that WB-RTO cancels TCP's inability to administer simultaneous retransmissions and consequently WB-RTO achieves higher goodput, better fairness and less retransmission overhead.

Table 1
TCP-RTO performance

	Goodput	Retransmissions
1st flow	1.95 KB/s	600 packets
2nd flow	2.2 KB/s	620 packets
3rd flow	1.8 KB/s	550 packets
4th flow	2.0 KB/s	600 packets
5th flow	2.05 KB/s	620 packets
Total	10.0 KB/s	2990 packets

2. Related work

Several researchers have reported problems regarding the TCP-RTO [5,7,19–21,8,17,9,22,23,10]. Lixia Zhang, in [10], identifies several drawbacks of the TCP retransmission timer and reports its intrinsic limitations. The paper concludes that mainly external events should trigger retransmissions and timers should be used only as a last notification about packet loss. Although WB-RTO departs from a different point, it also alleviates problems reported in [10], due to the fact that these problems are mainly caused by the exclusive relation of the timers with the RTT.

The Eifel Algorithm [7,8,6] focuses on spurious timeouts. A spurious timeout happens in case of a sudden delay spike on the link, where the round-trip delay exceeds the expected value calculated for the retransmission timeout. As a result, all data in flight are falsely considered lost and are being retransmitted. The Eifel algorithm uses the TCP timestamp option [24] to detect spurious timer expirations.

Once a spurious timeout is detected, the sender does not back-off, but instead, it restores the transmission rate to the recorded rate, prior to the timeout event.

The Forward RTO algorithm [9] targets the detection of spurious timeouts too. The algorithm, instead of using timestamp options, checks the ACK packets that arrive at the sender after the timer's expiration. F-RTO observes whether the ACKs advance the sequence number or not, concludes whether the timeout was spurious or not and determines the appropriate recovery policy, accordingly.

Both the above algorithms (Eifel [8] and F-RTO [9]) improve TCP's performance [25,26] significantly and are currently in the standardization process of the *Internet Engineering Task Force* [27–29]. However, none of them really solves the problems stated in [10], due to the fact that they do not modify the retransmission timeout algorithm itself, but instead they only change the *response* of the transport protocol *after a timeout has occurred*. More precisely, both algorithms ([8,9]) take into consideration outstanding data packets only after the timer expires, while the nature of the problem calls for design modifications of the timeout algorithm itself.

Recently, authors in [30] investigated the problem of TCP flow synchronization too and identified anomalies in TCP's transmission policy. More precisely, they report TCP performance problems in terms of fairness (i.e., TCP treats unfairly flows which experience higher RTTs) and window synchronization. They attempt to break flow synchronization by randomizing the sending times in TCP. *Randomized TCP* attempts packet transmissions with a time interval $\Delta = \mathrm{RTT}(1 + x)/cwnd$, where x is a zero mean random number drawn from a uniform distribution. The proposed scheme guarantees better fairness and reduced window synchronization. In the current work we target similar achievements (i.e., flow de-synchronization), but focus on the retransmission rather than on the transmission strategy of standard-TCP.

In [17], we have shown that timeout adjustments, based solely on RTT estimations, do not always correspond to the level of flow contention. We investigated the behavior of TCP in high-contention scenaria and confirmed that it is possible for the timeout to decrease when contention increases. We concluded that this anomaly is due to flow synchronization. Our analysis called for a new design that incorporates an estimation of the "degree" of contention, along with a mechanism to cancel synchronization. We exploit those directives in the present work.

3. WB-RTO: The proposed algorithm

We propose a new timeout algorithm which: (i) estimates the contribution of each flow to congestion (Section 3.1) *and* practically approximates the current level of network contention (Section 3.2) and (ii) allows for asynchronous retransmissions, ordered in time in reverse proportion to their contribution to congestion (Section 3.3). We note that (i) and (ii) form a collaborative detect/respond scheme to increase efficiency of the *system* response, i.e., not to optimize each flow's response: if all flows calculate a single accurate time for retransmission, the system will fail to provide efficient service when contention is high, due to flow synchronization. In our case, we detect the current network conditions (i.e., the level of flow contention) through mechanism (i) and schedule retransmissions accordingly through mechanism (ii), to provide the appropriate system response.

3.1. Proportional timeout

During high contention, it is possible for all flows to operate with minimal windows, in which case randomization guarantees timeout diversity. However, it is also possible for the contending flows to operate with different window sizes. In that case, we attempt to adjust the timeout according to the degree of the flow's contribution to congestion. Consequently, we introduce a methodology for estimating the contribution of each flow to congestion whenever some sender times out. In particular, we initially classify the flow depending on its current transmission window, charge it with an appropriate penalty, and hold the corresponding value in parameter c, according to Eq. (4):

$$c = f(cwnd\_, max\_cwnd\_), \tag{4}$$

where the current congestion window ($cwnd\_$) is compared with the maximum congestion window ($max\_cwnd\_$) that the flow has reached, since the last timeout expiration.[1] Frequent updates of the $max\_cwnd\_$ (here each time a timeout happens)

[1] The procedure is called every time the RTO is calculated for a specific TCP flow.

allow for better capturing of the network dynamics (i.e., joining and leaving flows).

- In case the *cwnd_* is smaller than half of the maximum congestion window, the flow is marked with the minimal charge ($c = 1$).
- If the flow's *cwnd_* belongs to the interval $[\frac{1}{2} \cdot max\_cwnd\_, \frac{3}{4} \cdot max\_cwnd\_)$, the penalty is higher ($c = 1{,}5$).
- Finally, if the current *cwnd_* lies in the remaining area (i.e., $[\frac{3}{4} \cdot max\_cwnd\_, max\_cwnd\_]$), the flow is given a major penalty ($c = 2$).

The justification for the above policy is as follows: the RTO is extended in proportion to the charge and, in turn, the flow waiting time to service is extended in proportion to its window. By the same token, flows that operated with small windows are rewarded with faster retransmission.

3.2. Contention estimation

Next, we further classify each flow according to its (recent) congestion window history (average window, denoted as *awnd_*). We define four different thresholds[2] (*Threshold 1–4*) that trigger different response upon a timeout event and classify each flow according to its *awnd_* value, which further corresponds to the current level of flow contention. $Threshold_1$ corresponds to very high contention (here set to 5 packets), while $Threshold_4$ refers to congestion events that happen sparsely (here set to 50 packets).

We assign four weights (a_1 to a_4) that correspond to the four predetermined intervals (0, $Threshold_1$) to ($Threshold_3$, $Threshold_4$):

$$a_i = g(awnd\_, Threshold_i), \tag{5}$$

where $a_i < a_{i-1}$. Next, a_i multiplies c in order to set the penalty and consequently the RTO, based on both the window and its history. *In conclusion, parameter c determines the contribution of each flow to contention, while the four predetermined Thresholds provide a rough categorization of the level of flow contention, currently, in the network. Both estimations are needed since none of them alone could guarantee both efficient link utilization and fair transmission scheduling: the first lacks the generic context of flow contention estimation, while the second cannot associate penalty policy with previous behavior.*

3.3. Timeout adjustments

We randomly select a value from the interval (rtt, $c \times a_i$), where the lower bound, rtt, guarantees that the timeout will not expire prior to the RTT, preventing the algorithm from becoming too aggressive. Each flow executes the following steps:

1. compares *cwnd_* with *max_cwnd_* and assigns a penalty accordingly (Eq. (4)).
2. estimates the level of contention, according to the flow's *awnd_* (Eq. (5)).
3. finally, calculates the Window-Based Retransmission Timeout.

$$\text{WB-RTO} = random(rtt, c \times a_i) \tag{6}$$

or

$$\text{WB-RTO} = random(rtt, f(cwnd\_, max\_cwnd\_) \times g(awnd\_, Threshold_i)). \tag{7}$$

Although sophisticated mechanisms, like bandwidth estimators, may provide self-adjustable values, we note that in the present work both the *Thresholds* and the corresponding a values are set experimentally.

3.4. Behavior of the algorithm

In Fig. 3, we present the behavior of WB-RTO for a wide range of average transmission windows. Three plots are presented in this figure. Each line plot represents the response of WB-RTO in relevance with the average transmission window for the three possible penalties. There are three salient points to make: (i) the highest values (in average) for the retransmission timeout correspond to the highest penalties, (ii) it is possible for a flow to calculate a small RTO even when it operates with large windows, (iii) timeout settles to smaller values as the average window increases. Note that in all cases, we prevent the algorithm from calculating a timeout value smaller than the round trip time, avoiding an aggressive behavior, which would negatively impact system performance.

[2] Note that both the thresholds and the parameters discussed in this work are determined based on experiments and they constitute subject of further investigation (see Section 5). Results presented in this paper are based on the values discussed below, unless it is explicitly stated otherwise.

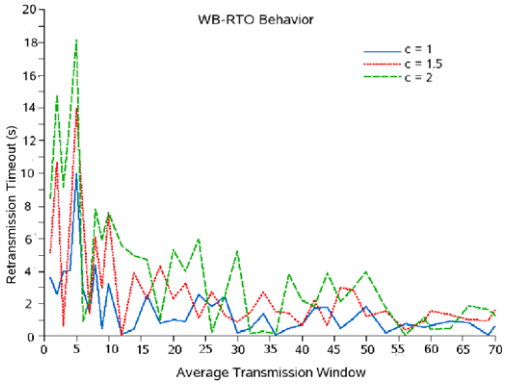

Fig. 3. WB-RTO vs. awnd_.

4. Performance evaluation plan

We have implemented our evaluation plan on the ns-2 network simulator [31]. We implemented the *Window-Based Retransmission Timeout* in TCP-Reno and thus, we use the same congestion control and error recovery algorithms as in TCP-Reno (i.e., slow-start, fast retransmit and fast recovery). We expect that, with the exception of Tahoe, which lacks the Fast Recovery mechanism, other TCP versions (e.g. SACK [32]) yield similar comparative results, as well.

Our evaluation plan involves the following three steps:

1. In the first step (Section 5), we evaluate the impact of the pre-determined values (i.e., a_1, a_2, a_3, a_4) that correspond to different Thresholds (i.e., $Threshold_1$ to $Threshold_4$), where the *Thresholds* are set to 5, 10, 30 and 50, respectively and the corresponding parameters (a_1, a_2, a_3, a_4) are set to 10, 5, 3 and 2, respectively.
2. At the second step (Section 6), we discuss the statistical validity of results produced by a randomly generated, uniformly distributed seed, which corresponds to timeout values chosen by the flows (Eq. (6)). Hence, we choose a different random seed for each flow, repeat the experiment 30 times and present the variation of the results. Since this is proved to be insignificant, we used the same seed for all flows throughout the experiments.
3. At the third step (Section 7), we evaluate the performance of the proposed algorithm. For that purpose, we choose scenarios which correspond to:
   - different propagation delay (e.g. wired vs. satellite, lossy links, see Section 7.1).
   - Active Queue Management schemes (e.g. RED, see Section 7.2).
   - traffic diversity (e.g. mice together with elephants, see Section 7.3.1 and dynamic network conditions, see Section 7.3.2).

Apart from the simulations presented in Sections 7.3 and 7.1.2, all others last for 1500 s, a time-period that deemed appropriate to allow all protocols demonstrate their potential. In addition, all flows enter the system randomly within the first two seconds of the simulation, in order to avoid initial flow synchronization. We use the topologies shown in Fig. 4, where the buffer size is set to be equal to the Delay-Bandwidth Product (DBP) of the outgoing link. Furthermore, in case of RED [33] we set the minimum and maximum thresholds to 1/10th and 3/10ths of the buffer size respectively, in accordance with [33].

We use two traditional performance metrics:

- the system Goodput, defined as:

$$\text{Goodput} = \frac{Original\_Data}{Connection\_time}, \tag{8}$$

  where *Original_Data* is the number of Bytes delivered to the high-level protocol at the receiver (i.e., excluding the retransmitted packets and the TCP header overhead) and *Connection_time* is the amount of time required for the data delivery.
- the system fairness, measured by the Fairness Index [34], defined as:

$$\text{Fairness} = \frac{\sum(Throughput_i)^2}{n\sum(Throughput_i^2)}, \tag{9}$$

  where $Throughput_i$ is the Throughput of the ith flow and n is the number of the participating flows.

Throughout the experiments we also measure (i) the goodput achieved per flow, in order to justify the fairness performance of the protocols, (ii) the number of retransmitted packets, in order to weight the overhead produced by each participating flow and (iii) the queue length, which indicates how efficiently the two algorithms exploit network resources. Whenever deemed necessary, we use the sequence number progress as a criterion to characterize the responsiveness of each algorithm.

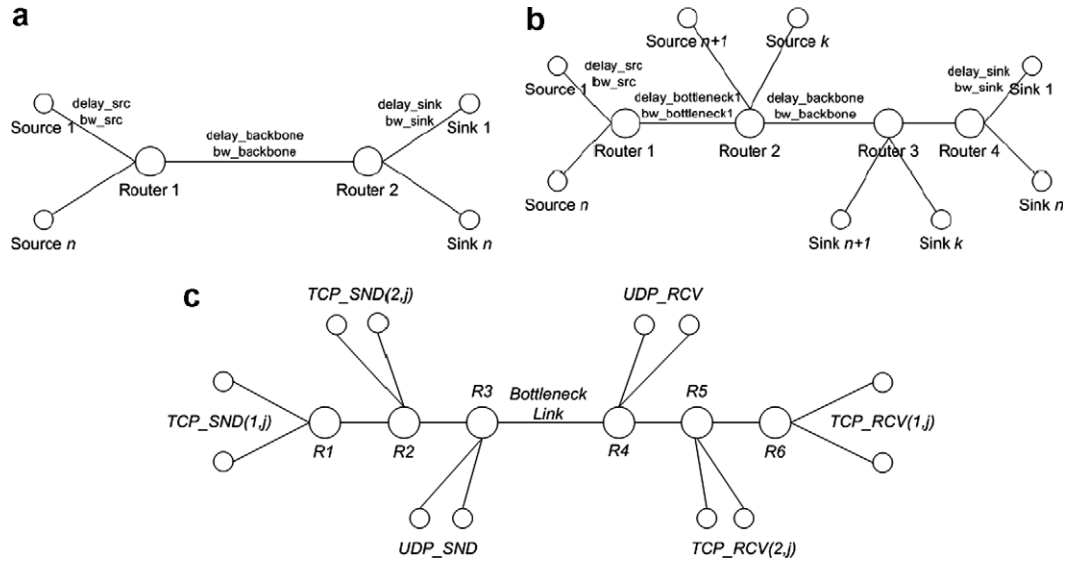

Fig. 4. Simulation topologies: (a) Dumbbell network topology, (b) Cross-traffic network topology, (c) Extended cross-traffic network topology.

5. Impact of pre-determined values

We present two significantly different scenarios (a satellite and a wired scenario) to evaluate the fixed values assignment of parameters a_1, a_2, a_3 and a_4. For this purpose, we choose three sets of values, which correspond to three distinct scales called Small Scale,[3] Medium Scale[4] (this is also the default set of parameters presented in Section 3 and is used in the rest of the paper) and Wide Scale,[5] respectively. It is obvious that the larger the values of the parameters assigned to the WB-RTO, the larger the possibility of an extended retransmission timeout value, since the interval $(rtt,\ c \times a_i)$ becomes larger.

5.1. Satellite scenario

We have observed that (see Tables 2 and 3 for scenario setup and simulation results, respectively) by assigning large values (i.e., Wide Scale modification) to the retransmission timeout a flow is possibly forced to continuously wait for long periods before it tries to re-enter the channel. On the contrary, the Small Scale modification, leads to aggressive retransmission attempts without improving the overall system performance.

We notice that the default parameters (Medium Scale) achieve higher Goodput performance and reduced retransmission overhead compared to the

Table 2
Simulation parameters

Satellite scenario setup	
Topology	Dumbbell
bw_src = bw_sink	10 Mbps
delay_src = delay_sink	10 ms
bw_backbone	5 Mbps
delay_backbone	350 ms
Queue policy	RED
Buffer size	200 pkts
Number of flows	150
Simulation time	1500 s

Table 3
Simulation results

Protocol	Performance		
	Goodput (KBytes/s)	Fairness	Retransmitted packets
TCP-RTO	571.248	0.999	97,816
WB-RTO			
Small scale	587.275	0.997	61,155
Medium scale	588.686	0.995	53,878
Wide scale	587.738	0.991	45,570

[3] $a_1 = 5$, $a_2 = 3$, $a_3 = 2$ and $a_4 = 1.5$.
[4] $a_1 = 10$, $a_2 = 5$, $a_3 = 3$ and $a_4 = 2$.
[5] $a_1 = 20$, $a_2 = 10$, $a_3 = 5$ and $a_4 = 3$.

TCP-RTO. The Small Scale modification results in more timeout expirations and consequently retransmits approximately 9% more packets than the Medium Scale does (i.e., 7500 packets), without improving Goodput. On the contrary, the Wide Scale modification retransmits even less packets than the default WB-RTO (i.e., Medium Scale), but results in unfairer behavior than the Medium Scale modification, although one would expect fairer behavior instead, especially in case of high contention. However, there is a fragile balance between timeout values and fairness, which Wide Scale does not seem to always administer well. Specifically, a sender that is waiting for long time prior to retransmitting may end up having a small(er) window at the next congestion event. According to WB-RTO algorithm, the misinterpreted situation calls for long timeouts again, and repeatedly, may lead to actual exclusion from utilizing the link. We conclude that the Medium Scale modification provides both fair and efficient transmission scheduling.

5.2. Wired scenario

The scenario setup for the next experiment is shown in Table 4 and the results in Table 5. Again, we observe that the Small Scale modification, as well as the standard TCP-RTO result in aggressive retransmission attempts and in turn in unfair retransmission scheduling. The Wide Scale modification now exhibits better fairness but reduced retransmission effort in the cost of system Goodput performance.

WB-RTO with the Medium Scale modification transmits successfully approximately 8 packets (8KB) per second (2% of the overall Goodput) less than TCP-RTO or WB-RTO with the Small Scale modification. However, both the above algorithms appear to be relatively unfair (5–20%). The unfair behavior is caused by the aggressive retransmission policy, which entails a large number of unnecessary retransmissions (see number of retransmitted packets in Table 5). For example, standard-TCP transmits 100 packets per second more than the Medium Scale WB-RTO.

Using the ratio $\frac{\text{Successful Transmissions}}{\text{Retransmissions}}$ as a criterion of the appropriateness of our settings, we conclude that the Medium Scale modification is a better choice for a default setting.

However, sophisticated mechanisms for bandwidth estimation (e.g. FAST-TCP [35]) may be used to dynamically adjust the Scale according to particular scenarios.

Table 4
Simulation parameters

High congestion scenario setup	
Topology	Dumbbell
bw_src = bw_sink	5 Mbps
delay_src = delay_sink	10 ms
bw_backbone	5 Mbps
delay_backbone	30 ms
Queue policy	Drop tail
Buffer size	19 pkts
Number of flows	150
Simulation time	1500 s

Table 5
Simulation results

Protocol	Performance		
	Goodput (KBytes/s)	Fairness	Retransmitted packets
TCP-RTO	598.187	0.634	223,411
WB-RTO			
Small scale	598.103	0.793	88,042
Medium scale	590.835	0.828	68,370
Wide scale	581.958	0.852	53,407

6. Statistical observations

The Window-Based RTO algorithm includes execution of a function, which randomly chooses values for the retransmission timeout. In this section we evaluate the contribution of the seed (i.e., randomly selected) to the statistical deviation of the mean and we find it to be insignificant (less than 1%). We repeated our experiments 30 times and captured the corresponding maximum, minimum and mean values, for each one of the metrics (goodput, fairness, retransmitted packets). The details of the statistical data for the Satellite and the Wired scenario are shown in Tables 6 and 7, respectively.

Each table includes also the corresponding results of TCP-RTO and WB-RTO using the same seed (Raw Seed). The *Difference* field presented in the tables is calculated according to Eq. (10):

$$\text{Difference} = \frac{|\text{Raw Seed Value} - \text{Random Seed Value}|}{\text{Raw Seed Value}} \times 100. \quad (10)$$

Table 6
Simulation results

Protocol	Performance		
	Goodput (KBytes/s)	Fairness	Retransmitted packets
TCP-RTO	571.248	0.999	97,816
WB-RTO			
Raw seed	588.686	0.995	53,878
Random seed			
Minimum	587.460	0.994	53,804
Difference (%)	0.208	0.1	0.137
Maximum	588.539	0.995	54,418
Difference (%)	0.024	0	0.992
Mean	587.932	0.995	54,115.4
Difference (%)	0.128	0	0.438

Table 7
Simulation results

Protocol	Performance		
	Goodput (KBytes/s)	Fairness	Retransmitted packets
TCP-RTO	598.187	0.634	223,411
WB-RTO			
Raw seed	590.835	0.828	68,370
Random seed			
Minimum	588.767	0.787	64,853
Difference (%)	0.35	4.95	5.144
Maximum	592.949	0.885	72,147
Difference (%)	0.35	6.44	5.235
Mean	590.689	0.839	68488.3
Difference (%)	0.024	1.311	0.172

7. Results

We evaluate the performance of WB-RTO under the following circumstances:

- In Section 7.1 we change link propagation delay: we use scenarios with wired and satellite links, respectively.
- In Section 7.2 we apply the RED queuing policy. We investigate the response of WB-RTO, when Active Queue Management is present.
- In Section 7.3 we, initially, mix mice with elephants and then apply dynamic network conditions (i.e., joining and leaving flows).

7.1. Performance of WB-RTO over wired and over satellite backbone links

7.1.1. Performance evaluation over a wired backbone link

The Delay-Bandwidth Product ($D \times B$) of the backbone link (see Fig. 4(a)) equals 10 packets and the router's Drop Tail buffer can hold up to 50 packets.[6] The simulation is repeated 10 times, increasing each time the number of participating flows $(10, 20, \ldots, 100)$. We have intentionally designed the simulated conditions so that resource supply does not suffice for what users demand. The purpose is to give the timeout algorithm the ultimate role of the transmission scheduler for the link. In our case, the main target of the retransmission timer is to distribute flows in time and permit *all* flows to utilize the network resources, instead of rejecting some flows for the benefit of the rest. Consequently, we expect improvement in terms of fairness.

Fig. 5 summarizes the performance of the two algorithms. WB-RTO slightly outperforms TCP-RTO in all cases, in terms of Goodput (Fig. 5(a)). The retransmission effort of the two protocols, however, differs dramatically.

One may naively think that the difference in Goodput performance achieved by WB-RTO is negligible compared to the traditional TCP-RTO. However, we point out that the *Retransmission* Timeout Algorithm is responsible for the *Retransmission* effort of the protocol, rather than for the actual Goodput performance of the protocol. Hence, we pay more attention on the combination of the *retransmission effort* spent by the protocol in order to achieve the measured *Goodput* performance, rather than on the Goodput performance alone.

In this context, we notice that WB-RTO outperforms TCP-RTO significantly in terms of fairness (see Fig. 5(c)); when contention increases (e.g. more than 60 flows) the scheduling property of the timeout becomes dominant. In effect, TCP-RTO behaves unfairly to some flows, since it continuously fails to provide a time scale for the whole system of flows, which could guarantee efficient link utilization. To strengthen our claims, we analyze the achieved Goodput per flow in Fig. 5(d), in case of

[6] According to [36] a large percentage of Internet flows have very low transmission rate (less than 100 KB/s). We try to capture such conditions in the current scenario.

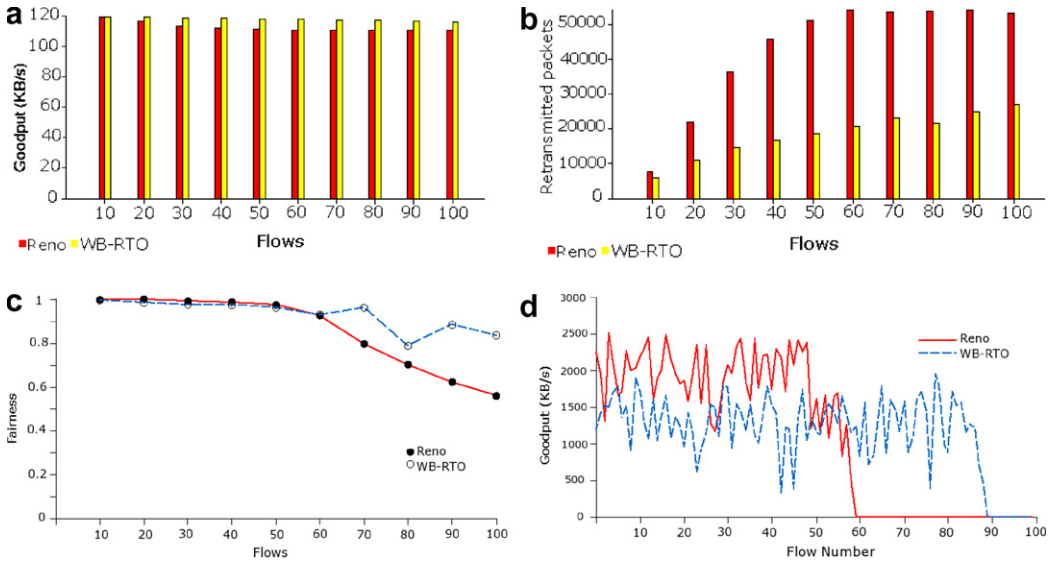

Fig. 5. Protocol performance in the dumbbell topology: (a) Goodput (in KBytes/s), (b) Retransmitted packets, (c) Fairness, (d) Goodput per flow (in Bytes/s).

100 participating flows. We observe that the same group of flows always (re-)transmits successfully (flows 1–60), while the rest of the flows (flows 61–100) repeatedly get rejected and achieve zero Goodput. On the contrary, WB-RTO guarantees resources to most of the participating flows (only 10 flows get rejected) and therefore distributes capacity in accordance with the demand. (Fig. 5(d)). We note that WB-RTO exhibits one significant property. It occasionally behaves aggressively and occasionally more conservatively, adjusting to the level of flow contention.

In order to verify our statements further, we present the bottleneck queue behavior for the above simulation. In case of TCP-RTO (Fig. 6(a)), we observe that although only 60 out of 100 flows compete for the backbone link (the rest get rejected), the buffer is always full. Hence, we conclude that the TCP retransmission policy is, in this case, rather aggressive: the time-slot allocated for retransmissions is too short. On the contrary, the conservative adjustments of the Window-Based RTO, which reflect a larger time scale for retransmissions, lead to more efficient buffer utilization (Fig. 6(b)).

7.1.2. Performance evaluation with a satellite backbone link

In this section we evaluate the performance of the Window-Based Retransmission Timeout in satellite environments. We use the topology of Fig. 4(b) where the satellite link has a propagation delay of 300 ms, and 20 Mbps bandwidth capacity (see Fig. 4(b)). All buffers use DropTail, except for the bottleneck buffer, which uses RED and holds up to 200 packets, with the *min* and *max thresholds* set at 20 and 60 packets, respectively and in accordance with [33]. All flows enter the system randomly within the first two seconds of the experiment.

The scenario involves also a 10^{-4} packet error rate and we additionally include occasional blackouts which impact the comparative Goodput performance of the algorithms (see Fig. 7). This is justified as follows: due to low contention, WB-RTO does not extend its scale much, permitting the TCP flows to exploit bandwidth faster (see progress of sequence number in Fig. 8) once it becomes available (i.e., right after the blackout). TCP-RTO instead, does not incorporate contention and hence its response corresponds to a falsely interpreted situation of congestion, extending the timeout unnecessarily. We explore the details of this scenario in Fig. 8. The arrow in Fig. 8(a) denotes the point in time where bandwidth becomes available but TCP-RTO fails to exploit it. The extra delay results in less Goodput (26,6% in case of 12 flows, see Fig. 7(b)).

More precisely, in Fig. 8, we notice that standard-TCP successfully transmits its last packet (before the blackout) and timeouts approximately one second later (because of ACK loss during the blackout). At this point it retransmits unsuccessfully, but this time it increases its timeout value exponentially, *misinterpreting the timeout event as*

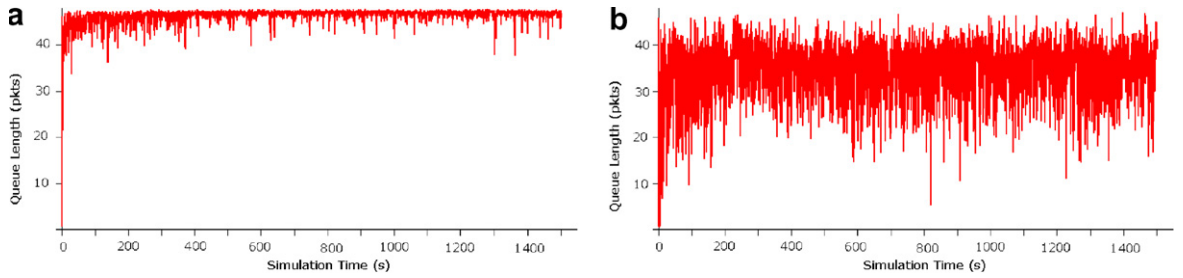

Fig. 6. Queue behavior: (a) TCP-RTO, (b) WB-RTO.

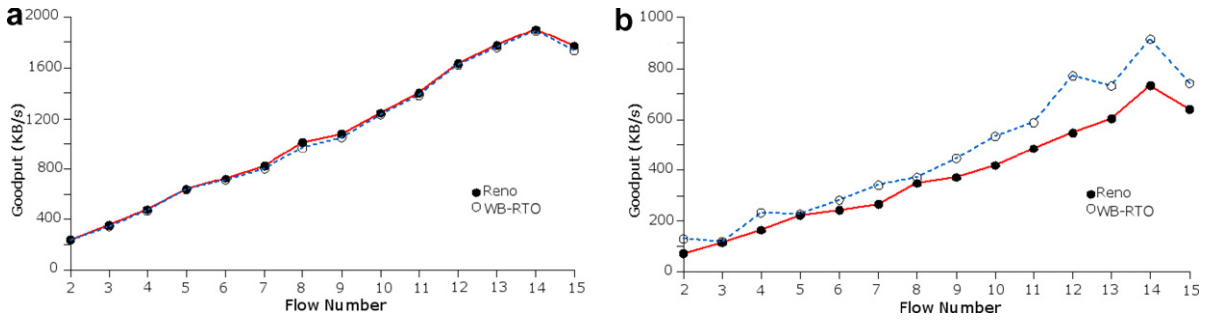

Fig. 7. Goodput performance: (a) No blackout, (b) After three blackouts in 150 s.

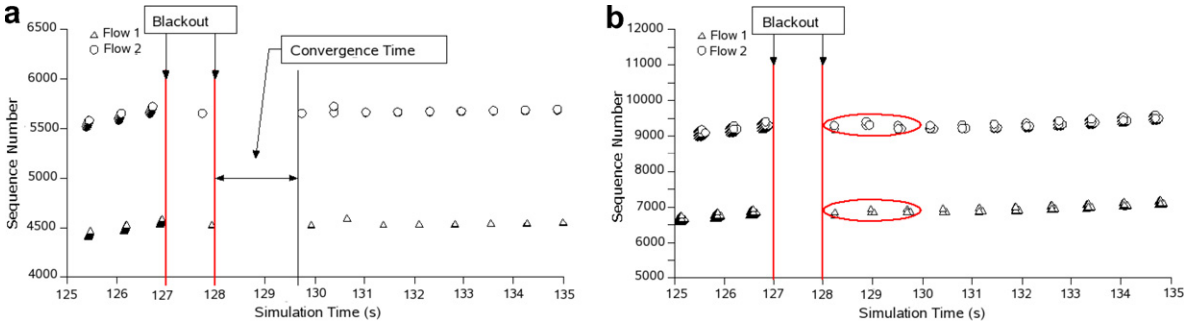

Fig. 8. Convergence time after the blackout: (a) TCP-RTO, (b) WB-RTO.

a signal of high contention. On the contrary, WB-RTO times out only once (because of ACK loss), but it does not extend the timeout further, since Eqs. (4) and (5) did not indicate high level of flow contention. Hence, we observe that WB-RTO resumes transmission almost immediately after the end of the blackout and successfully proceeds to the next transmission rounds, while TCP-RTO still waits for the second timeout to occur, due to false estimation of high contention.

7.2. WB-RTO with RED

The bandwidth of the backbone link is set to 10 Mbps and the propagation delay is set to 30 ms (Fig. 4(a)). We set the capacity of the RED gateway to 36 packets, according to the DBP, and configure the *min* and *max* thresholds to 4 and 12 packets, respectively.

Fig. 9 summarizes the performance of the two protocols. Although the Goodput performance (Fig. 9(a)) of TCP-RTO balances with WB-RTO, the associated effort spent by the protocols differs significantly. In particular, we observe in Fig. 9(c) that TCP-RTO retransmits 50% more packets than WB-RTO. For example, in case of 100 flows, TCP-RTO retransmits approximately 135 packets per second more, which is twice the effort expended by WB-RTO. We observe that the failure of TCP-RTO becomes more serious as contention grows. This observation is supportive to our argument that TCP needs to adjust its timeout to network

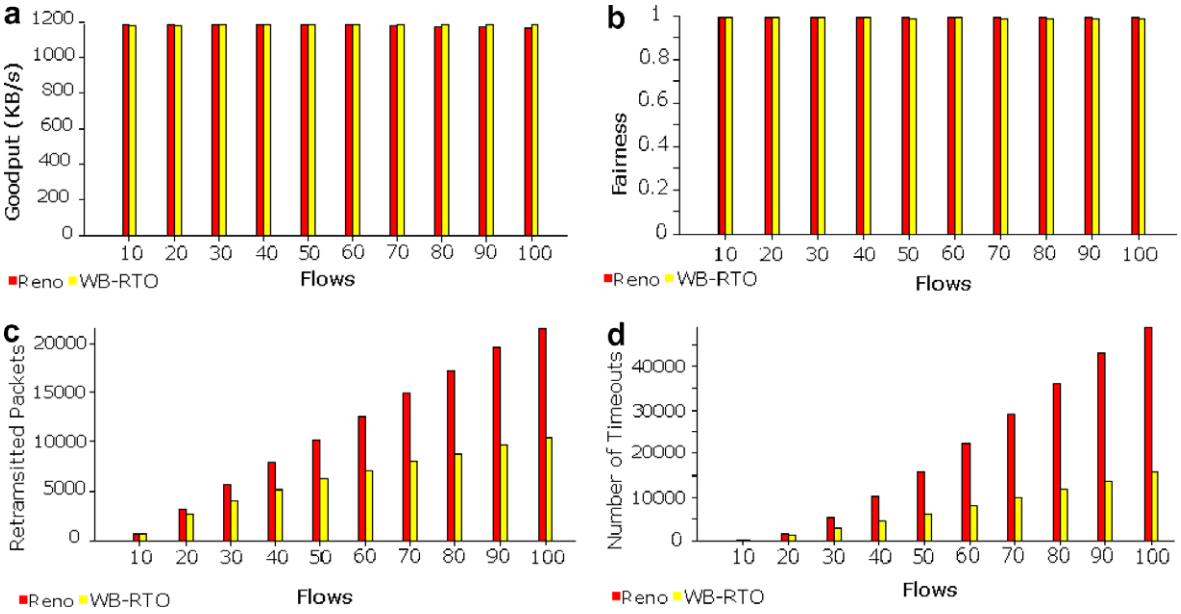

Fig. 9. Performance over high congested dumbbell backbone link: (a) Goodput (in Bytes/s), (b) Fairness, (c) Retransmitted packets, (d) Number of timeouts occurred.

contention. WB-RTO adjusts its scale progressively to network contention, widening the difference from standard-TCP. Consequently, in the worst case reported in Fig. 9(d), contention caused by 100 flows results in 66% less timeouts for WB-RTO.

Figs. 10 and 11 justify the same argument from another perspective. That is, the aggressive policy of TCP-RTO causes inefficient queue utilization, even when Active Queue Management schemes, namely RED, are present. More precisely, Fig. 10 shows that the queue length, in case of TCP-RTO, fluctuates a lot, overcomes the *maximum* threshold of the RED gateway and results in forced packet drops. We observe this behavior from the plot of the average queue length of the RED gateway in Fig. 11. Note that the average queue length for TCP-RTO is always above the *maximum* threshold, which in this experiment is set to 12 packets. On the contrary, WB-RTO rarely exceeds the *maximum* threshold of the RED gateway, (see Fig. 11(b)).

7.3. Traffic diversity

7.3.1. Mice together with elephants

We investigate the possibility of negative impact of WB-RTO on short flows. Due to the policy of "punishing" further flows that operate with smaller windows, one may consider that short flows (mice, such as Web-applications), which traditionally operate with small windows, may suffer further (e.g. "World-Wide-Wait" [37]). However, as we show in Fig. 12, the results do not support this claim. It appears that the dominant behavior of the WB-RTO algorithm is its capability to reduce the num-

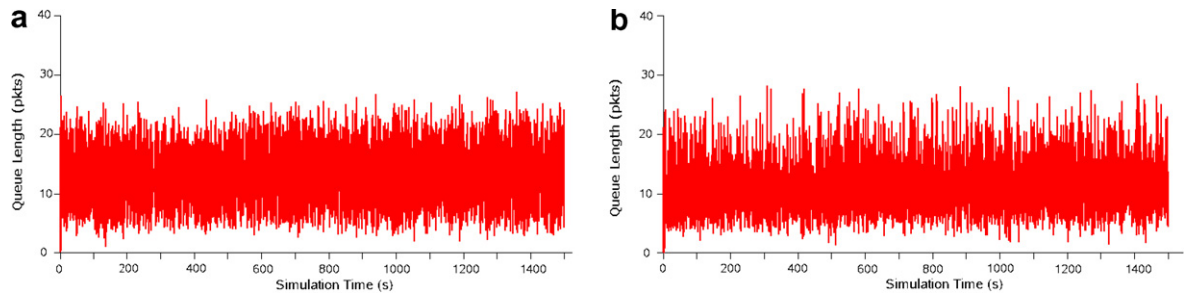

Fig. 10. Queue behavior: (a) TCP-RTO, (b) WB-RTO.

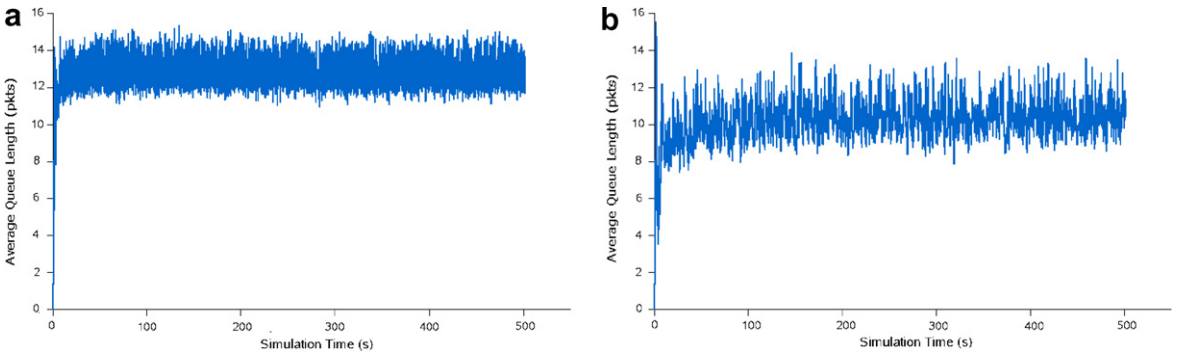

Fig. 11. Average queue length: (a) TCP-RTO, (b) WB-RTO.

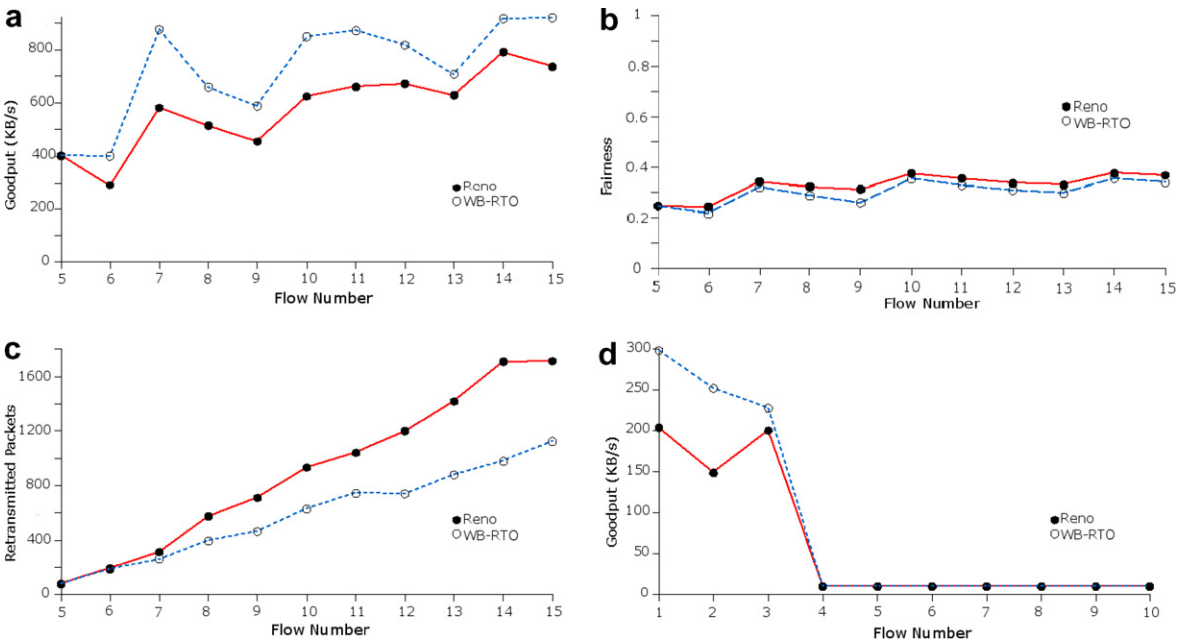

Fig. 12. Performance including mice: (a) Goodput (KB/s), (b) Fairness, (c) Retransmitted packets, (d) Goodput per flow (KB/s).

ber of timeouts. The impact of less timeouts is clearly more beneficial to mice than to elephants since the timeout, as a value, constitutes a larger portion of unexploited communication time for short flows. Hence, whatever side-effect may be of concern, it is canceled by the reduction of the timeout events.

According to [38], most of the traffic accommodated in Internet links is generated by a small number of long flows (elephants), while the vast majority of Internet connections carry short flows (mice), which occupy the rest of the Internet capacity.

We set up the scenario as follows: each time 70% of the participating flows run Web applications (i.e., they represent short flows), while the remaining 30% of the connections belong to long FTP applications. This means that in case of 15 participating flows, for example, only 4 sessions perform bulk data transfer. The DBP of the backbone link (see Fig. 4(a)) equals 40 packets, while its buffer, which acts according to the RED queuing policy, has a capacity of 40 packets as well. The short sessions are configured to transmit a 10 KB file (e.g. a text-based Web page), every second.

In Fig. 12 we see that WB-RTO greatly outperforms TCP-RTO both in terms of Goodput (Fig. 12(a)) and in terms of retransmission effort (Fig. 12(c)). In case of 14 participating flows, for instance, we see that WB-RTO transmits approximately 150 KB/s more than standard-TCP, and at

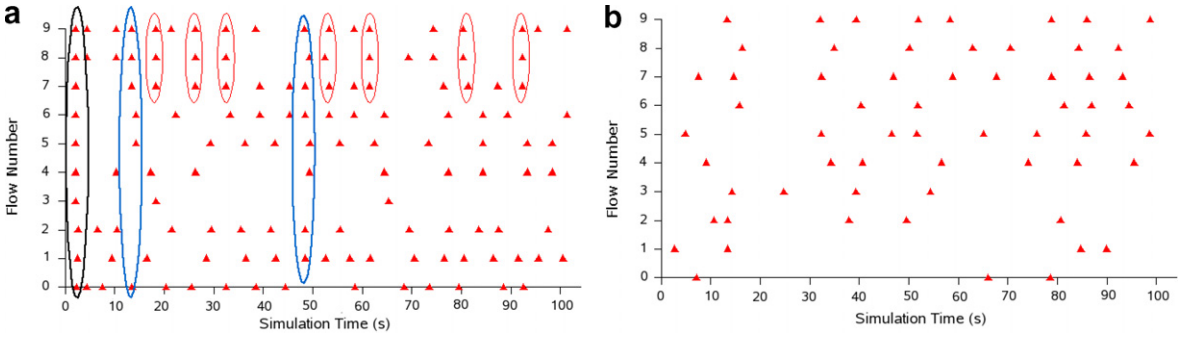

Fig. 13. Timeout events: (a) TCP-RTO, (b) WB-RTO.

the same time retransmits 80 packets (per second) less than standard-TCP, making a great amount of network resources available to possible incoming flows. In this context, we argue that the degraded goodput performance of standard-TCP is due to bad retransmission scheduling, which results in flow synchronization.

We observe simultaneous timeout expirations experienced by different TCP-RTO flows (see circled dots in Fig. 13(a)). With TCP-RTO, all 10 flows timeout simultaneously during the Slow-Start phase (first circle in Fig. 13(b)) and most of the timeouts experienced by TCP-RTO are spurious (WB-RTO does not expire in the same deterministic scenario). Although both the Forward RTO [9] and the Eifel [8] algorithms may detect the spurious timeout expiration and recover appropriately, they do not manage to de-synchronize TCP flows.

More interestingly, we notice that the last 3 short flows (i.e., flows 7, 8, 9) timeout simultaneously 10 times during the experiment (see circled dots in Fig. 13(a)). In addition, long standard-TCP flows time out quite frequently, unlike WB-RTO which timeouts sparsely and at random times. In particular, we count 83 timeout expirations for the short TCP-RTO flows, but only 50 for the WB-RTO. The situation gets even worse if we count the long flows' timeout expirations. That is, standard-TCP long flows timeout 46 times in total, while WB-RTO flows timeout only 12. This leaves more space to the long flows and minimizes the impact on the short flows' performance (Fig. 12).

7.3.2. Dynamic network conditions

We evaluate the performance of the proposed algorithm in case of dynamically changing network conditions. We use the topology of Fig. 4(c) and define five different groups of flows (four TCP and one UDP): $TCP\_SND_{i,j}$ nodes send endless data to $TCP\_RCV_{i,j}$ nodes, where $i,j = (1,2)$; 10 UDP_SND nodes send a 500 Bytes data packet every 10 ms to 10 UDP_RCV nodes. Each TCP group of flows consists of $\frac{n}{4}$ flows; we repeat the experiment ten times, increasing each time the number of TCP flows (i.e., $n = 10, 20, \ldots, 100$). All links in Fig. 4(c) have 100 Mbit/s bandwidth capacity, apart from the bottleneck link, which has 5 Mbit/s bandwidth capacity. The entering and leaving times of the respective groups of flows are shown in Fig. 14.

We observe (see Fig. 15) that although, occasionally, TCP-RTO slightly outperforms WB-RTO in terms of Goodput performance, it appears to be rather inefficient in terms of retransmission scheduling. See for example, the case of 20 flows in Fig. 15(a). TCP transmits successfully 3 KB/s more than WB-RTO, but expends effort for retransmitting 20 KB/s more than WB-RTO (Fig. 15(c)). As contention increases, however, we see that WB-RTO outperforms TCP both in terms of Goodput performance and retransmission overhead. When 80 flows compete, for example, WB-RTO transmits successfully 3 KB/s more than TCP-RTO and retransmits 50 KB/s less than TCP-RTO. Furthermore, WB-RTO improves fairness in all cases (Fig. 15(b)).

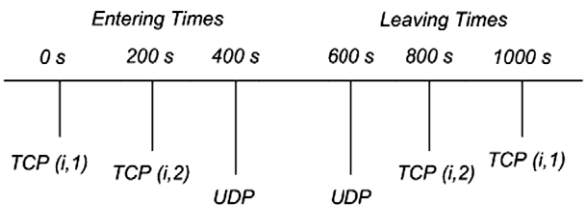

Fig. 14. Entering–leaving times.

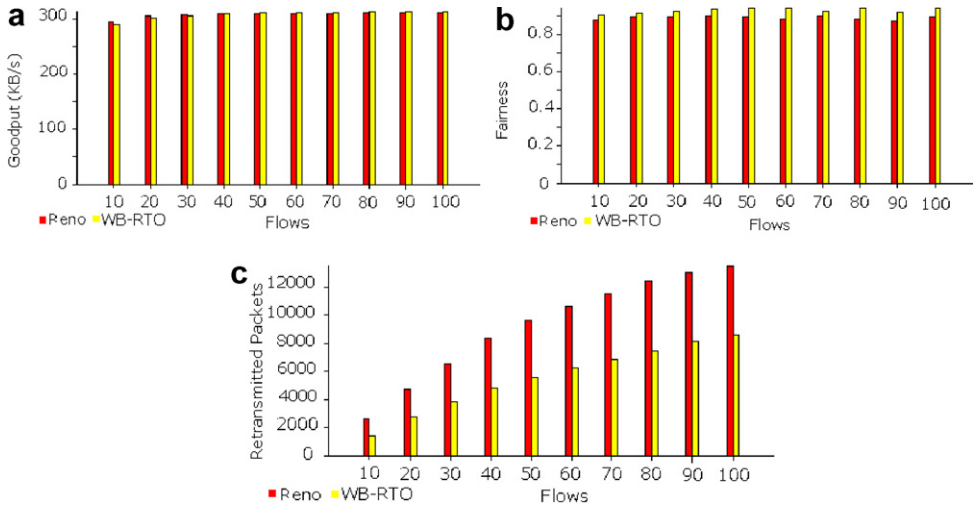

Fig. 15. Protocol Performance, Dynamic Scenario: (a) Goodput (Bytes/s), (b) Fairness, (c) Retransmitted packets.

We have also traced the sequence number progress for the flows of the above experiment, although we do not present it here, due to space limitations. We observed that WB-RTO, in accord to its design principles (see Eqs. (4) and (5)), assigns longer retransmission timeout *penalties* to flows with higher transmission rates. That is, the *cwnd_* of flows with higher transmission rate will fluctuate closer to the *max_cwnd_* value and hence, Eq. (4) will assign greater penalty to such flows. In particular, WB-RTO extended the timeout of the $TCP_{2,j}$ group of flows canceling somewhat the $TCP_{1,j}$ group's comparative disadvantage due to larger propagation delay.

8. Conclusions

We designed WB-RTO, based on the following principles:

1. When contention is high, the timeout algorithm becomes the scheduler for the link and, hence, it has to adjust its scale accordingly.
2. A flow should schedule its retransmission early or late, depending on its contribution to current contention.
3. Randomization of timers should be designed in a manner that the link is fully utilized but synchronization is avoided.

We evaluated the performance of the proposed algorithm and found that our design principles match well our simulation results. We observed significant improvement in retransmission effort and fairness. We have shown that WB-RTO applies to various network conditions, such as different network delays, link heterogeneity (lossy links, occasional blackouts), Active Queue Management schemes and traffic diversity.

References

[1] J. Postel, Transmission Control Protocol, RFC 793, 1981.
[2] V. Paxson, M. Allman, Computing TCP's Retransmission Timer, RFC 2988, May 2000.
[3] P. Karn, C. Partridge, Improving round-trip time estimates in reliable transport protocols, in: Proceedings of ACM SIGCOMM, 1987.
[4] P. Sarolahti, A. Kuznetsov, Congestion control in linux TCP, in: Proceedings of USENIX'02, June 2002.
[5] H. Ekstrom, R. Ludwig, The Peak-Hopper: a new end-to-end retransmission timer for reliable unicast transport, in: Proceedings of IEEE INFOCOM, 2004.
[6] A. Gurtov, R. Ludwig, Evaluating the Eifel algorithm for TCP in a GPRS network, in: Proceedings of European Wireless, 2002.
[7] A. Gurtov, R. Ludwig, Responding to spurious timeouts in TCP, in: Proceedings of IEEE INFOCOM, 2003.
[8] R. Ludwig, H. Katz, The Eifel algorithm: making TCP robust against spurious retransmissions, ACM Computer Communication Review (2000).
[9] P. Sarolahti, M. Kojo, K. Raatikainen, F-RTO: an enhanced recovery algorithm for TCP retransmission timeouts, in: Proceedings of ACM SIGCOMM, 2003.
[10] L. Zhang, Why TCP timers don't work well, in: Proceedings of ACM SIGCOMM, 1986, pp. 397–405.
[11] M. Allman, V. Paxson, On estimating end-to-end network path properties, in: Proceedings of ACM SIGCOMM, 1999, pp. 263–274.

[12] E. Blanton, M. Allman, On making TCP more robust to packet reordering, ACM Computer Communication Review 32 (2002).
[13] P. Benko, G. Malicsko, A. Veres, A large-scale, passive analysis of end-to-end TCP performance over Gprs, in: Proceedings of IEEE Infocom 2004, Hong Kong, China, 2004.
[14] M. Scharf, M. Necker, B. Gloss, The sensitivity of TCP to sudden delay variations in mobile networks, in: Proceedings of NETWORKING 2004, Athens, Greece, 2004, pp. 76–87.
[15] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, D. Towsley, Measurement and classification of out-of-sequence packets in a tier-1 IP backbone, in: IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet Measurement, New York, NY, USA, ACM Press, 2002, pp. 113–114.
[16] J.C.R. Bennett, C. Partridge, N. Shectman, Packet reordering is not pathological network behavior, IEEE/ACM Transactions on Networking 7 (1999) 789–798.
[17] I. Psaras, V. Tsaoussidis, L. Mamatas, CA-RTO: a contention-adaptive retransmission timeout, in: Proceedings of ICCCN, 2005.
[18] V. Tsaoussidis, I. Matta, Open issues on TCP for mobile computing, The Journal of Wireless Communications and Mobile Computing, WCMC 2 (2002).
[19] E. Blanton, M. Allman, Using TCP Duplicate Selective Acknowledgement (DSACKs) and Stream Control Transmission Protocol (SCTP) Duplicate Transmission Sequence Numbers (TSNs) to Detect Spurious Retransmissions, RFC 3708, 2004.
[20] Y.C. Kim, D.H. Cho, in: Considering Spurious Timeout in Proxy for Improving TCP Performance in Wireless Networks, Computer Networks, vol. 44, Elsevier Science, 2004, pp. 599–616.
[21] Y. Seami, K. Le, Decorrelated Loss Recovery (DCLOR) Using SACK Option for Spurious Timeouts draft-swami-tsvwg-tcp-dclor-06, 2004.
[22] S. Fu, M. Atiquizzaman, Modeling TCP Reno with spurious timeouts in wireless mobile environments, in: Proceedings of ICCCN 2003, Dallas, USA, 2003.
[23] V. Paxson, End-to-end internet packet dynamics, in: Proceedings of the ACM SIGCOMM'97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, Computer Communication Review, vol. 27 (4), ACM Press, 1997, pp. 139–154.
[24] V. Jacobson, R. Braden, D. Borman, TCP Extensions for High Performance, RFC 1323, May 1993.
[25] M. Allman, V. Paxson, W. Stevens, TCP Congestion Control, RFC 2581, 1999.
[26] W. Stevens, TCP slow start, congestion avoidance, fast retransmit, and fast recovery algorithms, RFC 2001, 1997.
[27] R. Ludwig, M. Meyer, The Eifel Detection Algorithm for RFC 3522, in preparation.
[28] R. Ludwig, A. Gurtov, The Eifel Response Algorithm for TCP, RFC 4015, 2005.
[29] P. Sarolahti, M. Kojo, Forward RTO-Recovery (F-RTO): An Algorithm for Detecting Spurious Retransmission Timeouts with TCP and the Stream Control Transmission Protocol (SCTP), RFC 4138, 2005.
[30] K. Chandrayana, S. Ramakrishnan, B. Sikdar, S. Kalyanaraman, On Randomizing the Sending Times in TCP and other window based algorithms, Computer Networks, vol. 50, Elsevier Science, 2006, pp. 422–447.
[31] ns 2: The Network Simulator – ns – 2. http://www.isi.edu/nsnam/ns/.
[32] M. Mathis, J. Mahdavi, S. Floyd, A. Romanow, TCP Selective Acknowledgement Options, RFC 2018, 1996.
[33] S. Floyd, V. Jacobson, Random early detection gateways for congestion avoidance, IEEE/ACM Transactions on Networking 1 (1993) 397–413.
[34] D.M. Chiu, R. Jain, Analysis of the increase and decrease algorithms for congestion avoidance in computer networks, Computer Networks and ISDN Systems 17 (1989) 1–14.
[35] D.X. Wei, C. Jin, S.H. Low, S. Hegde, FAST TCP: motivation, architecture, algorithms, performance, IEEE/ACM Transactions on Networking, in press.
[36] K. Lan, J. Heidemann, A measurement study of correlations of internet flow characteristics, Computer Networks 50 (2006) 46–62.
[37] R. Khare, I. Jacobs, W3C Recommendations Reduce 'World Wide Wait'. http://www.w3.org/Protocols/NL-PerfNote.html.
[38] L. Guo, I. Matta, The war between mice and elephants, in: Proceedings of ICNP 2001.

Ioannis Psaras received a diploma in Electrical and Computer Engineering from Demokritos University of Thrace, Greece in 2004, where he is currently a Ph.D. candidate. He won the Ericsson Award of Excellence in Telecommunications for his diploma dissertation in 2004. Ioannis has worked, as a research intern at DoCoMo Eurolabs (May–September 2005) and at Ericsson Eurolab (May–September 2006). His research interests lie in the area of transport protocols and their behavior over heterogeneous (wired, wireless, satellite, high-speed) links.

Vassilis Tsaoussidis received a B.Sc. in Applied Mathematics from Aristotle University, Greece; a Diploma in Statistics and Computer Science from the Hellenic Institute of Statistics; and a Ph.D. in Computer Networks from Humboldt University, Berlin, Germany (1995). Vassilis held faculty positions in Rutgers University, New Brunswick, SUNY Stony Brook and Northeastern University, Boston. In May 2003, he joined the Department of Electrical and Computer Engineering of Demokritos University, Greece. His research interests lie in the area of transport/network protocols, i.e. their design aspects and performance evaluation. Vassilis is editor in chief for the Journal of Internet Engineering and editor for the journals IEEE Transactions in Mobile Computing, Computer Networks, Wireless Communications and Mobile Computing, Mobile Multimedia and Parallel Emergent and Distributed Systems. He participated in several Technical Program Committees in his area of expertise, such as INFOCOM, NETWORKING, GLOBECOM, ICCCN, ISCC, EWCN, WLN, and several others.