CA-RTO: A Contention-Adaptive Retransmission Timeout

I. Psaras, V. Tsaoussidis, L. Mamatas
Dept. of Electrical and Computer Engineering
Demokritos University of Thrace
Xanthi, Greece
Email: {ipsaras, vtsaousi, emamatas}@ee.duth.gr

***Abstract*— We show that TCP timers, based solely on RTT estimations and measurements, cannot capture with precision the level of flow contention. We notice that increased contention may stabilize RTT variation, minimize the deviation and, in turn, shorten the timeout. We show that this behavior is undesirable indeed, since it leads to unfair resource utilization. We propose CA-RTO, an algorithm that incorporates a contention parameter and a randomization technique into the Retransmission Timeout. We report significant improvement in fairness, great reduction of retransmitted packets and slight improvements in application goodput.**

I. Introduction

Statistical Multiplexing, the main technology of packet networks, allows for bandwidth sharing on demand and cancels the possibility for flow starvation. However, the level and type of multiplexing is mainly determined by the transmission policies of the flow senders - that is, the sum of all sender transmissions determines current demand. Since transmission adjustments are based on detected network conditions, resource demand fluctuates in accord. Therefore, the application rates are not really reflected in the multiplexer but rather this is determined by the transmission policy of the transport protocol. Fair multiplexing therefore is clearly becoming an issue of how fairly demand is currently reflected by the transport protocol current windows or rates, and in turn, how accurate and simultaneous is the network-condition detection by the participating flows.

When the resource demand exceeds the supply, that is, when flow contention grows, multiplexing exhibits different characteristics. For example, since packets are dropped due to high demand and limited supply, current demand, and consequently the strategy for multiplexing is determined mainly by the timeout algorithm. That is, the timeout becomes the scheduler for the link.

Although the design of the timeout algorithm has been studied extensively in the past [5], [8], [9], [13], its association with link scheduling has not. In addition, its scheduling properties have not really been evaluated adequately. Instead, much attention has been paid on its ability to reflect network delay accurately, 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 really capture network contention. Hence, it is possible for contention to grow, while the timeout shrinks, at least for those flows which experience less delays in the queue. Specifically, flows with large windows, a common situation when flow contention is low, do capture network delay with precision: last packets of the same window will experience more delays. The situation differs when each flow is represented with a single-packet window, for example.

The problem of scheduling as it is associated with timeout has another dimension as well. When flows are synchronized, or becoming synchronized due to a congestion event, the timeout is adjusted accordingly for all participating flows. The existing policy to exclude the retransmitted (dropped) packets from measurements leaves little space for timeout differentiation among the participating flows, thus leading to possibly synchronized retransmissions. Therefore, fairness cannot be guaranteed since flows are not randomly transmitted, but instead, are possibly partitioned into two or more groups: the ones that utilize the link and the others that attempt to enter at times when the link is utilized.

Flow synchronization does not appear only when flows enter the system simoultaneously. In addition, since congestion is a single and common point in time for all flows, it may cause synchronization by itself. The situation is similar to cars queuing on a red sign and restarting together. The degree of synchronization here actually depends on the diversity of the timeout, the level of contention and the capacity and algorithm of the buffer; these factors determine how many flows will have one or more of their packets dropped.

We depart from exploiting the aforementioned situation with simulations. We confirm that the timeout as a scheduler may be problematic indeed. Next, we attempt to introduce randomness into the timeout algorithm in order to solve the aforementioned synchronization problem and the associated fair resource utilization problem. We achieve that by integrating a contention-adaptive parameter into the timeout. Finally, we confirm that different levels of contention call for distinct timeout adjustments. As a side-effect, we realize that link goodput may also be improved. Driven by encouraging results, we analyze the behavior of the new algorithm in detail [1]. We show that a contention-adaptive timeout (i) may improve significantly fairness (ii) has the potential to break synchronization due to timeout and (iii) reduce amount of sender retransmission and

[1] given the space limitations

therefore it may also improve system goodput.

We used the following structure: In section II we summarize the related work. In section III we present the timeout algorithm currently deployed in TCP; we also discuss the problems associated with the lack of contention-based adjustments. We propose our solution in section IV; discuss our evaluation methodology in section V; and present our results and their justification in section VI. We draw the main conclusions in section VII.

II. Related Work

Many researchers have reported TCP's timer problems. Lixia Zhang in [13] states that timers have intrinsic limitations in offering optimal performance: any timer is a "guess-based" scheduler. She concludes that high performance TCP should be based on external notifications upon forthcoming failures and that a timer should be used only as an auxiliary indication. More recently, researchers focused on the fact that the timeout algorithm is RTT-dependend and study the RTT variations in order to optimize the timeout value. Most of the research that takes place in that field targets the avoidance of *Spurious Timeouts*. For example, if an RTT measurement increases that much to exceed the value of the retransmission timeout that had been determined prior to packet transmission, the timeout expires although data is still in flight. In this context, Ludwig and Katz introduce the Eifel Algorithm [8] in order to get aware of spurious timeouts. The Eifel algorithm uses the timestamp options defined in [6]. Every retransmitted packet includes a unique timestamp. The sender compares this timestamp with the one that the corresponding ACK carries. If the ACK's timestamp is smaller, then this obviously indicates a spurious timeout; the Eifel algorithm restores the connection state saved before the timeout instead of backing off.

Another approach in the same context is presented in [2]. In the event of a timeout the sender retransmits and measures the RTT until the corresponding ACK arrives. If this RTT is smaller than the minimum RTT measured so far, then this indicates that the ACK was already in flight when the sender retransmitted (spurious timeout).

In [7] the authors propose two ways to improve TCP throughput in wireless networks. The first way is to select a timeout value higher than the standard one. In this way timeout events are reduced. The second way is the technique of selective repeat (SR) and go-back-N (GBN) policies upon timeout expirations.

III. Timeout Scheduling Requires Contention-Based Adjustments

TCP [10], transmission policy is confined by the rules of congestion control [1], [12], which is based, in principle, on the Additive Increase-Multiplicative Decrease (AIMD) [3] algorithm[2]. Due to TCP's overriding transmission rules, application rates (demand) may be reformed at the transport layer and consequently at the link layer as well.

[2]AIMD is coupled in TCP with several other optimizing mechanisms

TCP's transmission rules, however, dominate the transmission policy only when congestion is not really catastrophic; when demand (contention) exceeds the supply significantly, the timeout undertakes a more dominant role.

The timer is adaptive to varying delay: it is calculated every RTT after smoothing out the measured samples, and weighting recent history. Therefore, the timeout function as a projection of expected network delay. More precisely, given a sample measurement M and a history of average RTT A, the distance of the average is measured:

$$Diff = M - A,$$

the average is updated

$$A = A + gDiff,$$

and the deviation is calculated as:

$$Dev = Dev + d(|Diff| - Dev).$$

Finally, the Retransmission Timeout is adjusted to

$$RTO = A + 4D,$$

where A is an estimator of the average RTT (smoothed RTT) and D is the smoothed mean deviation. $Diff$ is the difference between the measured and the estimated RTT. Factor g is set to 0.125 and d is set to 0.25 [9]. A higher value for d causes RTO to respond faster to RTT variations.

The above equations do not allow for differentiation among different flows that experience the same queuing delay. For example, 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 buffer is fully utilized, may also be excluded in the next rounds as well, for the same reason. Current timeout scheduling becomes very deterministic, allowing only a particular set of participating flows to utilize the link.

Clearly, the existing RTO formula depends on the *average (smoothed) RTT* and on the *smoothed mean deviation* alone. However, based on simulations, we claim that contention increase cancels the deviation gradually, and in turn, reduces the RTO value falsely. Therefore, although deviation is indeed an optimizing addition to RTO when contention is low, it exhibits a contradicting role when contention grows. For example, when the network is not congested, transmission windows grow, resulting in large RTT variation within the same window. Suppose a congestion window of 50 packets. The 1^{st} packet of the window occupies the 1^{st} position in the router's buffer, while the 50^{th} packet occupies the 50^{th} position. The queing delay will obviously have impact on both the RTT and the *smoothed mean deviation*. Now suppose that 100 flows compete for 1Mbps backbone link. Under such conditions, TCP senders operate with small windows,

leading to minimal deviation within the same window. Given a possible synchronization, deviation is also minimal between the different rounds; not only between different packets within the same window. Suppose a TCP sender sends a window of 5 packets. Arriving at the router these packets will either get dropped or occupy the last positions in the router's buffer. In this case, the RTT will not vary, but instead it will continuously get the larger value measured for this connection[3]. Since there is no RTT variation, the *smoothed mean deviation* approaches 0. The RTO formula gives:

$RTO \approx A$, since $4D \rightarrow 0$.

The above analysis, is confirmed by the next set of simulations. We simulate a contention decrease scenario. Initially, until the 250^{th} second, we simulate 1 flow over a 1Mbps backbone link (dumbbell network topology). After the 250^{th} second 99 more flows enter the system.

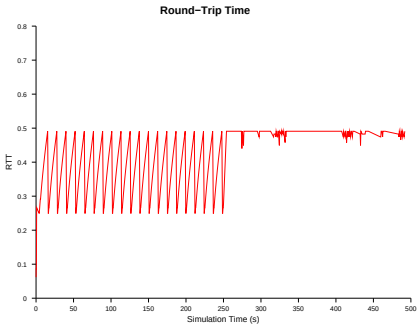

Fig. 1. RTT Measurement

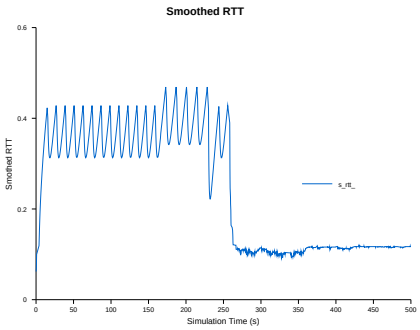

Fig. 2. Smoothed RTT ($RTO = \mathbf{A} + 4D$)

Figure 1 captures the RTT variation of the first flow we plotted however, similar graphs for the rest of the flows. We observe that RTT variation is canceled in case of high contention. Incoming packets (at the bottleneck buffer) either get dropped or occupy the last position in the queue. Cancelation of the RTT variation leads to a *Mean Deviation* that approaches 0 (Figure 3). Thus, the RTO (Figure 4) in this case depends

[3]the larger value corresponds to the last position in the router's buffer

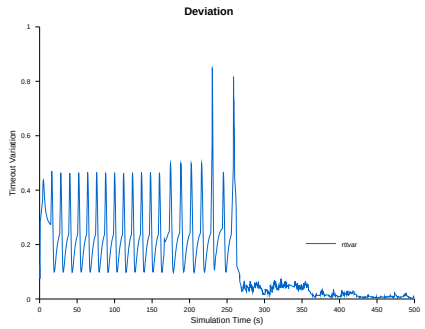

Fig. 3. Mean Deviation ($RTO = A + \mathbf{4D}$)

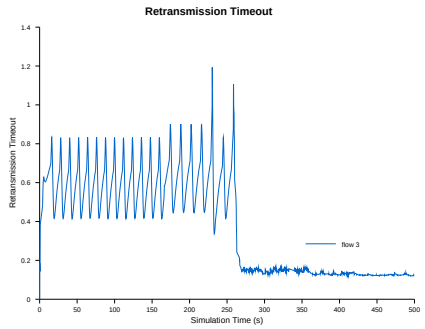

Fig. 4. Retransmission Timeout

only on the Smoothed Average RTT (Figure 2). Based on the above analysis for the timeout algorithm, we conclude that it does not behave in the desired manner when it becomes the sole link scheduler. Although there isn't link and buffer underutilization, due to the high contention, the above situation results in degraded system fairness, unnecessary timeout expirations and bad link distribution among the competing flows. We validate the above statements by extensive simulations and present the results in Section VI.

IV. CA-RTO: The Proposed Algorithm

A. The Design Goals

We propose an enhancement to the timeout formula, which makes the timeout *contention dependent*. The purpose of the CA-RTO algorithm (Contention-Adaptive Retransmission Timeout) is twofold:

1) **To incorporate contention.** The major design goal of the proposed algorithm is to *measure contention* and reflect it into the timeout by increasing its value in proportion.
2) **To introduce retransmission randomness.** The second design goal is to randomly adjust/extend the timeout value in case of high contention, in order to *break possible synchronization*.

We *measure contention* by means of the congestion window. More precisely, we employ the following idea:
During communication, the sender records the *max* congestion window ($max\_cwnd\_$) prior to congestion. When contention

increases, the sender may experience congestion with much smaller window. That is, it can detect contention, approximately, by measuring the difference between the current window and the max recorded window. The larger the difference, the higher contention is. We implement this as follows:

We introduce a factor c, where

$$c = \frac{1}{cwnd\_}$$

We measure contention difference as:

$$cont\_diff\_ = max\_cwnd\_ - cwnd\_$$

In order to *break synchronization*, we introduce factor p to adjust the temporal properties of retransmission as follows: When contention is increased the timeout which triggers retransmission is randomly selected from a wider range of distinct timeslots. When contention is low, the time scale adjusts back to a smaller range of distinct slots.

We achieve this by using the space $(0, cont\_diff\_)$ for calculating p:

$$p = Random(0, cont\_diff\_)$$

where $cont\_diff\_$ has been adjusted earlier to correspond to the timeout scale as:

$$cont\_diff\_ = cont\_diff\_ * 100\%$$

Finally, the RTO is calculated as:

$$RTO = A + 4D + c * p$$

B. CA-RTO Response To Contention

In case of static network conditions, p gets a small value and the product $c*p$ depends mostly on the c factor, which has minimal impact when contention is low. In case of dynamic network conditions (e.g. Contention Increase) p gets a larger value. That is, in case of Contention Increase factor c grows due to the large denominator, while factor p grows as well due to extended $cont\_diff\_$. As a result, $c*p$ product gets a large value. Thus, each flow randomly chooses a timeout value from the range of $(RTO, RTO + c * p)$

The following Figures 5, 6 monitor both the behavior of the $c * p$ product and of the proposed *Contention-Adaptive Retransmission Timeout*. The simulation scenario is the same as the one presented in Figures 1, 2, 3, 4. The impact of the $c*p$ product on the RTO value is clear. The proposed algorithm adjusts the timeout value according to the network conditions. The algorithm randomly extends the retransmission timeout value when the network is congested.

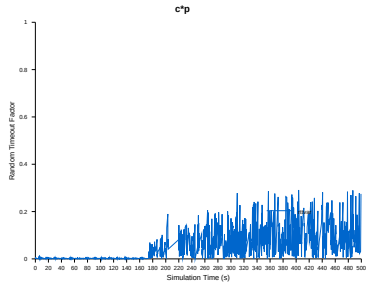

Fig. 5. c*p

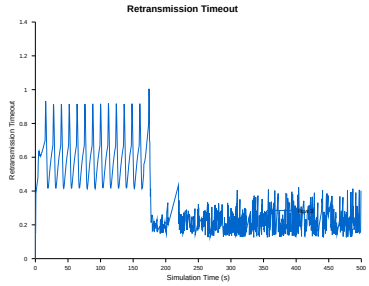

Fig. 6. Contention-Adaptive Retransmission Timeout

V. Evaluation Methodology

A. Simulation Testbed

We have implemented our evaluation plan on the ns-2 network simulator. The network topology used as a test-bed is the typical single-bottleneck dumbbell, as shown in Figure 7. The link's capacity (bw_bottleneck) is either 1Mbps or 10Mbps. The bw_src is 1Mbps, and the bw_dst is 0.5Mbps. We used equal number of source and sink nodes. We simulated an heterogeneous (wired and wireless) network with ns-2 error models which were inserted into the access links at the sink nodes. The Bernoulli model was used to simulate link-level errors with configurable packet error rate (PER). The number of flows occasionally changes for the different scenarios. The simulation time was fixed at 500 seconds, a time-period that seemed appropriate to allow all protocols to demonstrate their potential.

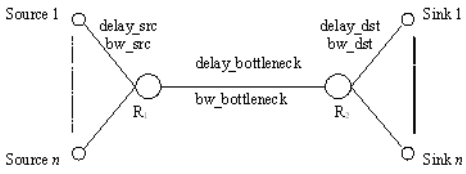

Fig. 7. Simulation Topology

In order to validate our statements, we selected and evaluated the TCP Reno protocol with both DropTail and RED queuing algorithms. Note that the *Contention-Adaptive Retransmission Timeout* is implemented in TCP-Reno. The buffer size is 50 packets, unless it is explicitly stated otherwise and

the *min* and *max* thresholds for the RED algorithm are set to 15 and 45 respectively [4].

VI. Simulation Results

We present a set of experiments that show the potential of CA-RTO in high-contention environments. We also show that the retransmission timeout value is not affected in case of low contention. In other words, the proposed algorithm is being activated only when it should be. In Figures 8, 9, 10, 11 the bw_bottleneck is 1Mbps and the buffer size is set to 50 packets.

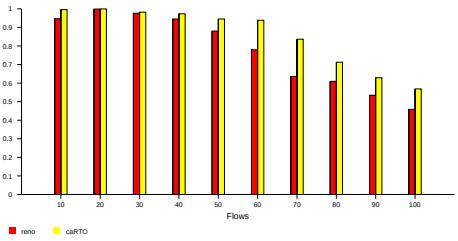

Fig. 8. Fairness

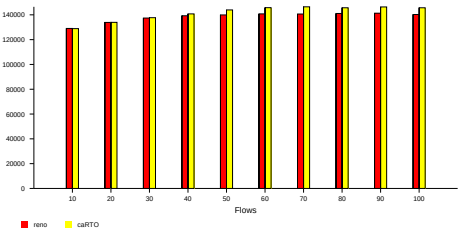

Fig. 9. Throughput (y axis measures Bps)

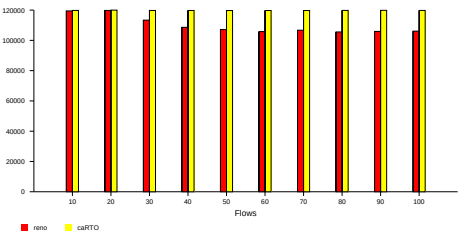

Fig. 10. Goodput (y axis measures Bps)

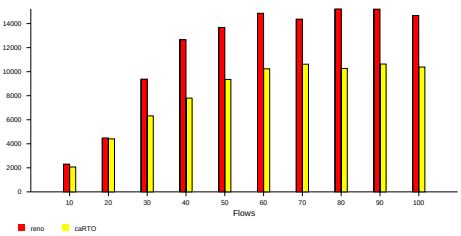

Fig. 11. Retransmitted Packets (y axis measures packets)

The results in Figures 8, 9, 10, 11 clearly show that a contention-adaptive retransmission timeout achieves fairer and more efficient link scheduling. More precisely, in Figure 8 we see that CA-RTO can achieve better fairness up to 0,2 Index Points (70 flows). Figure 9 shows that CA-RTO does not achieve major Throughput performance gains, due to the high number of retransmissions (Figure 11). We can observe that TCP-Reno makes approximately 4000 more (unnecessary) retransmissions (compared to CA-RTO), something that leads to increased Throughput performance due to large amount of overhead. However, this performance of standard TCP is not invested in Goodput.

The results are even more encouraging when contention further increases. In the following scenario (Figures 12, 13) the bw_bottleneck is 10Mbps and the buffer size is set to 100 packets. Figure 12, shows that CA-RTO's Fairness Index is always at least 0,15 points higher than the traditional TCP-Reno. CA-RTO has approximately 25% less retransmissions than TCP-Reno (Figure 13). That is, in this case about 4500 more retransmitted packets.

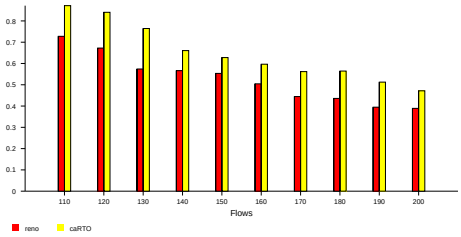

Fig. 12. Fairness

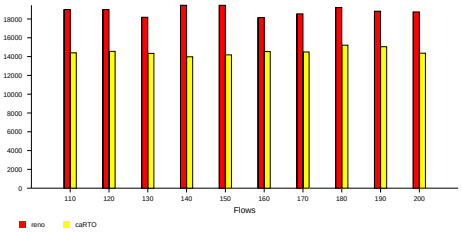

Fig. 13. Retransmitted Packets (y axis measures packets)

Next, we show that the proposed algorithm does not affect the timeout calculation when congestion events occur sparcely. The backbone link (bw_bottleneck) is 100Mbps. TCP senders grow their windows, since there is enough fair-share. In this case, factor c gets a very low value and consequently the CA-RTO algorithm avoids the timeout extension. Figure 14 shows that the difference in Goodput approches 0, in this case.

In Figure 15 the bw_bottleneck is 100Mbps, we use the RED algorithm and deploy a graduated contention decrease. All flows enter the system within the first two seconds. For the rest 498 seconds there is graduated contention decrease, starting from 10 flows and repeating the experiment for 20, 30, up to 70 flows. At each stage we reduce the number of

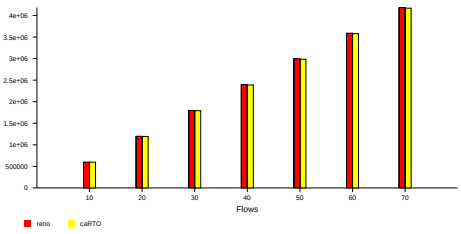

Fig. 14. Goodput (y axis measures Bps)

flows to half every *decrease step*, where *decrease step*, is the step needed in order for the last flow to exit at the 500^{th} second. Although contention decreases, RED forces packet drops, resulting in continuous timeout events. However, not all timeouts happen because of packet drops. By randomly extending the timeout value, CA-RTO avoids unnecessary timeouts leading to approximately 1700 less retransmitted packets (Figure 15, 60 flows).

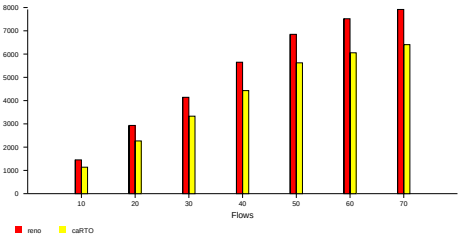

Fig. 15. Retransmitted Packets, RED

Finally, we simulate a wireless environment in order to monitor how CA-RTO responds in case of random errors. The bw_bottleneck is 1Mbps the buffer holds up to 150 packets and we inject 10% Packet Error Rate. The interesting point here is that although there is a rather high level of contention, the difference in the fairness index is not outstanding (as it was previously).

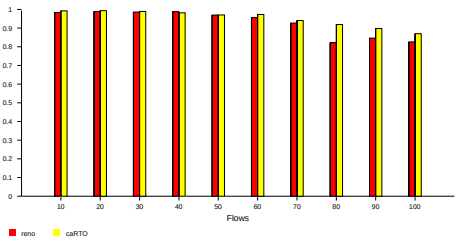

Fig. 16. Fairness

In this scenario, the performance difference between the two protocols is reduced (compared to previous results). As shown in [11], random link errors can break synchronization themselves, leading in this way to fairer link utilization.

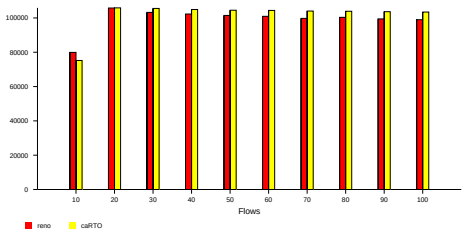

Fig. 17. Goodput (y axis measures Bps)

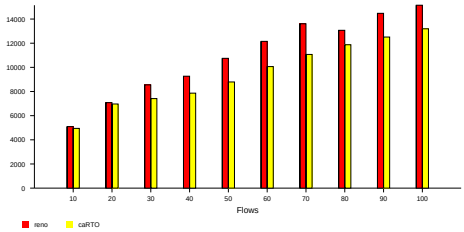

Fig. 18. Retransmitted Packets (y axis measures packets)

VII. Conclusions

We measure Goodput, Throughput, Fairness and Retransmitted Packets with a new algorithm, the CA-RTO (Contention-Adaptive RTO). The algorithm proved to have a twofold impact: it satisfied the major goal to improve fairness and achieved a somewhat unexpected result: it improved goodput as well. Although unexpected (i.e. originally was not a design goal) the result is also well-justified. Timeout randomization caused less unsuccessful retransmission attempts, thus allowing for less overhead and hence better link utilization.

Acknowledgment

This work was funded by the European Commission and the project PENED 2003 of GSRT.

References

[1] M. Allman, V. Paxson, and W. Stevens (April 1999) ”TCP Congestion Control”, RFC2581.
[2] M. Allman, V.Paxson (September 1999) ” On Estimating End-to-End Network Path Properties”, ACM SIGCOMM 1999.
[3] D.-M. Chiu and R. Jain (1989) ”Analysis of the Increase and Decrease Algorithms for Congestion Avoidance in Computer Networks”, Computer Networks and ISDN Systems, 17(1):1-14, 1989.
[4] S. Floyd, and V. Jacobson (August 1993) ”Random Early Detection gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, 1(4):397-413.
[5] A. Gurtov, R. Ludwig ”Responding to Spurious Timeouts in TCP”, IEEE Infocom 2003.
[6] V. Jacobson, R. Braden, D. Borman (May 1992), ”TCP Extensions for High Performance, RFC 1323.
[7] Kin K. Leung, T. Klein, C. Mooney and M. Haner ”Methods to Improve TCP Throughput in Wireless Networks With High Delay Variability”
[8] R. Ludwig and H. Katz (January 2000) ”The Eifel Algorithm: Making TCP Robust Against Spurious Retransmissions” ACM Computer Communication Review.
[9] V. Paxson, M. Allman (November 2000) ”Computing TCP’s Retransmission Timer”, RFC 2988.
[10] J. Postel (September 1981) ”Transmission Control Protocol”, RFC 793.
[11] I. Psaras and V. Tsaoussidis ”Good and Bad Symptoms of TCP Over Wireless Links” Submitted.

[12] W. Stevens (January 1997) ”TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms”, RFC 2001.
[13] Lixia Zhang ”Why TCP Timers Don’t Work Well”