Fuzzy Logic Strategy of Prognosticating TCP's Timeout and Retransmission

Zhongwei Zhang[1], Zhi Li[1] and Shan Suthaharan[2]

[1] Dept of Mathematics and Computing
University of Southern Queensland
Toowoomba, QLD 4350, Australia
zhongwei, zhili@usq.edu.au
[2] Department of Mathematical Sciences
University of North Carolina at Greensboro
Greensboro, NC 27402, USA
ssuthaharan@uncg.edu

Abstract: The work presented in this paper is the design and implementation of an intelligent strategy using fuzzy logic technology to gauge the TCP timeout and retransmission value. The conventional algorithms, which are based on statistical analysis, perform in a marginally acceptable way for estimating theses two values. But they have been shown to be increasingly incapable of dealing with more complicated TCP traffic due to ignorance of traffic complexity. Fuzzy logic technology will be applied to estimate the TCP timeout and retransmission for the purpose of utilising artificial intelligence, combining knowledge about the network traffic and connection.

Keywords: TCP, timeout, retransmission, fuzzy logic.

1 Introduction

The size and the complexity of computer networks have grown in past years. To achieve an efficient and reliable transmission, some protocols inevitably need to handle complicated network traffics, and unexpected transmission losses. These problems usually are referred to as flow control and congestion control. The technologies of managing complex computer networks need to be more circumspect not only in the sending host and receiving hosts, but also the intermediate routers. One of these protocols is the Transmission Control Protocol (TCP) that has a responsibility of ensuring reliability.

Because TCP guarantees the reliable delivery of data, it retransmits each segment if an ACK is not received in a certain period of time. TCP sets this timeout as a function of the round trip time (RTT) it expects between the two ends of the connection. Unfortunately, given the range of possible RTTs between any pair of hosts in the Internet, as well as the variation in RTT between the same two hosts

over time, choosing an appropriate timeout value is not very easy [1]. To address this problem, TCP uses an adaptive retransmission mechanism.

An important improvement occurred in 1986 when a simple algorithm was developed. The idea is that every time TCP sends a data segment, it records the time. When an ACK for that segment arrives, TCP reads the time again, and then takes the difference between these two times as a *SampleRTT* . TCP then computes an *EstimatedRTT* as a weighted average between the previous estimate and this new sample.

The main problem with the simple algorithm is that it does not take the variance of the sample RTTs into account. An improved algorithm was then developed. For details about these conventional methods, refer to Section 2.

The conventional methods which have been used in the past years have largely relied on statistical data analysis technology. However, the study of using fuzzy logic to improve TCP performance has been proposed in [2] where a fuzzy logic controller has been used to selectively drop in the cells in an unspecified bit rate (UBR) service. This paper introduces a new method for calculating TCP's timeout and retransmission value.

This paper has been organized in four sections. Section 2 explain the TCP timeout and retransmission problem and survey the relevant strategies. In Section 3, a new approach is introduced based on fuzzy inference. The experiment results are presented in Section 4. In the last section, we conclude our paper. In addition we also outline some possible improvements for the future.

2 TCP Timeout and Retransmission

TCP is one of the most predominate transport layer protocols in the TCP/IP protocol suits. TCP provides a reliable transport layer, even though the service it uses (IP) is unreliable. Every piece of TCP data that gets transferred around the Internet goes through the IP layer at both end systems and at every intermediate router. One approach for providing reliability is for each end to acknowledge the data it receives from the other end. But data segments and acknowledgment can get lost. TCP handles this by setting a timeout when it sends data, and if the data isn't acknowledged when the timeout expires, it retransmits the data.

Fundamental to TCP's timeout and retransmission is the measurement of the round-trip time (RTT) experienced on a given connection. The RTT is dynamic over time, due to changes in routes and changes in network traffic, and TCP should track these changes and modify its timeout accordingly.

In the early days of the Internet, a simple algorithm was used to determine the timeout (RTO). This algorithm doubles an estimated RTT; while an estimated RTT is a weighted RTT and the previous estimated RTT. That is:

$$EstimatedRTT = \alpha \times EstimatedRTT + (1 - \alpha) \times SampleRTT \quad (1)$$

$$TimeOut = 2 \times EstimatedRTT \quad (2)$$

This algorithm has been proven to be rather conservative. It also has a flaw: RTT might not be the time difference between the first acknowledgment or the second with the data transmission. The Internet was suffering from high levels of network congestion at that time. An improved algorithm was introduced in 1987 [9]

to mitigate the causes of congestion. The algorithm can at best fix some causes of that congestion, but cannot completely eliminate it. Soon, another approach has was proposed to battle congestion [4]. TCP calculates the round-trip time and then uses these measurements to keep track of a smoothed RTT estimator and a smoothed mean deviation estimator. These two estimators are then used to calculate the next retransmission timeout value. The new algorithm folds the variance into the timeout calculation as follows:

$$Difference = SampleRTT - EstimatedRTT \tag{3}$$

$$EstimatedRTT = EstimatedRTT + (\delta \times Difference) \tag{4}$$

$$Deviation = Deviation + \delta(|Difference| - Deviation) \tag{5}$$

$$TimeOut = \mu \times EstimatedRTT + \phi \times Deviation \tag{6}$$

Where based on experience, δ is a fraction between 0 and 1, μ is typically set to 1 and ϕ is set to 4. Thus, when the variance is small, TimeOut is close to *EstimatedRTT* ; a large variance causes the Deviation term to dominate the calculation.

Apparently, the approaches of estimating the timeout and retransmission value are conventionally based on statistical analysis. In the next section, we propose a new approach [2, 3, 5] based on fuzzy logic. Nevertheless, the basis of the new approach of computing the timeout and retransmission value is the same as the Jacobson/Karns algorithm. More specifically, the new approach computing the next timeout value is also based on the two smoothed estimators.

3 A Fuzzy Approach of Estimating the Timeout Value

Fuzzy logic looks at the world in imprecise terms, in much the same way that our own brain takes in information [6, 7].

While considering the retransmission of a packet, it is important to estimate an appropriate value for TCP to retransmit a packet (RTO). A big estimated RTO value results in data transmission becoming idle and thus network resources being wasted, while a small estimated RTO value results in unnecessary retransmission. In order to keep the network running efficiently, we designed a fuzzy system to implement an RTO setting which is capable of tracking the trend of RTTs and quickly mastering the situation of the network according to previous RTTs.

We knew that the relationship between RTT and RTO is nonlinear. Fuzzy inferencing systems perform well in the nonlinear dynamic systems. The fuzzy system is designed as in Figure 1:

The input variables for the system contain:

(1) ΔRTT: the variance between the current RTT and the previous one;
(2) ΔRTO_RTT: the deference between the current RTO and the current RTT.

The output variable for the system contains:

ΔRTO: the variance between the next RTO and the current RTO.

Once we have the variance between the next RTO and the current one, it is straightforward to calculate the next RTO as follows:

$$RTP_{next} = RTO_{prev} + \Delta RTO \tag{7}$$

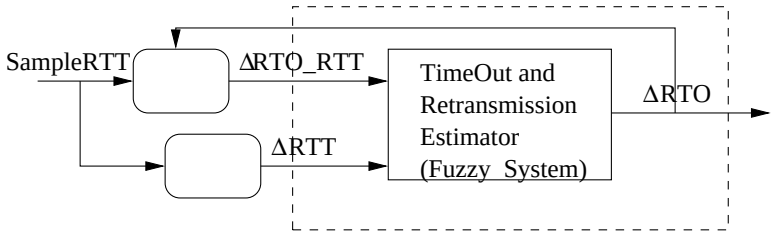

Fig. 1. Fuzzy inference system

Seven membership functions were used for each of the two inputs and for the output. The membership functions of ΔRTT defined fuzzy sets for

NB: Negative Big,
NM: Negative Medium,
NS: Negative Small,
ZERO: A fuzzy set,
PS: Positive Small,
PM: Positive Medium and
PB: Positive Big.

The membership functions of ΔRTO_RTT defined fuzzy sets for

VH: Very High,
H: High,
MH: Medium High,
M: Medium,
ML: Medium Low,
L: Low, and
VL: Very Low.

The seven membership functions for the output are

NB: Negative Big,
NM: Negative Medium,
NS: Negative Small,
ZERO: A special fuzzy set,
PS: Positive Small,
PM: Positive Medium and
PB: Positive Big.

Although the input and output are similarly labeled, the membership functions for the variables were independently specified and adaptable. The fuzzy estimator thus comprised a total of 21 linguistic variables. Triangular and trapezium shapes are adopted for both input and output membership functions. Triangular and trapezium shapes are adopted for both input and output membership functions. The distribution of membership functions are related with the average of RTT and RTO.

Figure 2 shows the membership functions of the input variables and output linguistic variables. Note that the memberships for ΔRTT and ΔRTO are symmetric, but the membership function for ΔRTO_RTT is asymmetric.

The design of the fuzzy linguistic rules for the inference of RTO takes into consideration the following conditions:

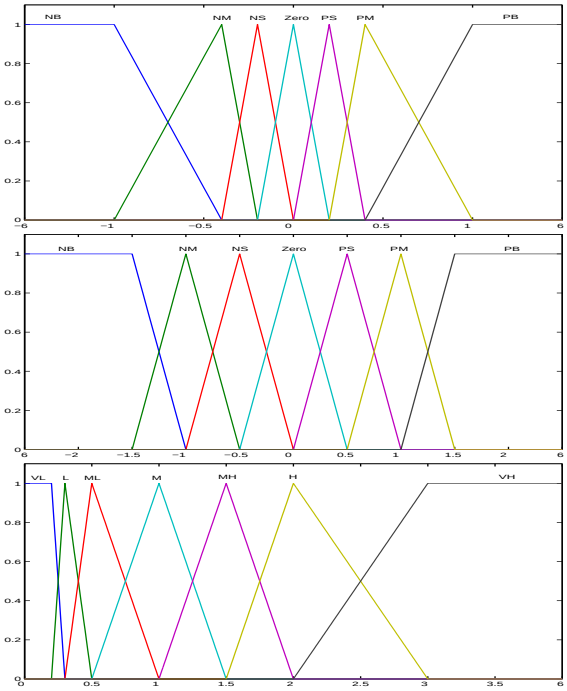

Fig. 2. Membership functions for the input and output variables

- The timeout is related to congestion. If you timeout too soon, you may unnecessarily retransmit a segment, which only adds to the load on the network.
- The timeout is related to the network situation. The previous RTTs are a good indicator of this; the network situation is well reflected by the average RTTs.
- The difference between the previous RTT and the mean RTT will contribute significantly to the current RTT accordingly.
- The current RTO is heavily reliant on the difference between the previous RTO and current RTT.

The fuzzy rules base consists of 7 rules, which are summarized in Table 1. For instance,

`if` Δ`RTT is NB OR` Δ`RTO_RTT is VH, then` Δ`RTO is NB`

The principle of the fuzzy rules is following the track of the RTT of data transmission and keeping the distance between the estimated RTO and practical RTT around an ideal time interval (1 second).

Min-product inference was used, along with center of gravity defuzzification. For more information about gravity defuzzification, refer to [6].

Table 1. Fuzzy rule base

#	ΔRTT	RTO-RTT	ΔRTO
1	NB	VH	NB
2	NM	H	NM
3	NS	MH	NS
4	ZERO	M	ZERO
5	PS	ML	PS
6	PM	L	PM
7	PB	VL	PB

4 Experiment Results

The fuzzy system has been tested on two different kinds of networks. It has been used to estimate the RTO on a real network and on a simulated network. With the simulated network, the fuzzy system has been used on a set of networks with different congestion levels.

4.1 A Real Network

An experiment can be carried out on the Internet or a computer network which constantly results in packet loss. We performed this experiment on our experimental network consisting of three Linux boxes named as maximian, Gordian and valerian. Basically, the timer for the connection is started when segment 1 is transmitted, and turned off when its acknowledgement (segment 2) arrives. The timer usually reports the clock ticks. We need to determine how many seconds the timeout value is. Figure 3 shows the relations between the actual RTT and the counted clock ticks.

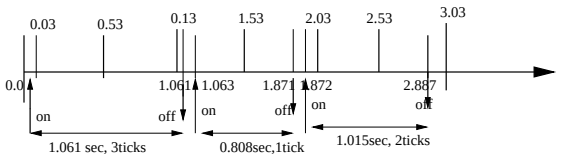

Fig. 3. Measuring the Timeout

The proposed fuzzy system has been implemented in MATLAB. The measured RTTs have been recorded by the tcpdump program [8]. In this experiment, 128 segments were transmitted, and 18 RTT samples were collected. Figure 4 shows the measured RTT along with the RTO used by TCP for the timeout. Note that the top one is calculated by using the Karn's algorithm and the middle one is calculated by the fuzzy estimator; while the bottom one are the measured RTTs.

This experiment has demonstrated that fuzzy inference can be applied in predicting the TCP timeout and retransmission value. Secondly, it is obvious that the RTO calculated by the fuzzy estimator is less than that calculated by the Karn's

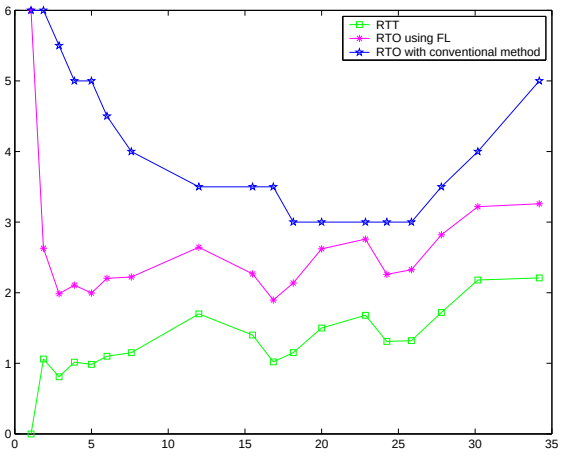

Fig. 4. MeasuredRTT and RTOs calculated

algorithm. It indicates that the RTO calculated by using fuzzy inference is finer. Also the fuzzy estimator can quickly get ready to predict the RTOs, meaning the RTO drops so quick that it gives a much better estimation within 3 seconds.

The calculated RTO by our fuzzy strategy is more sensitive to the dynamics of MeasuredRTTs, and RTO by the Karn's method is less sensitive to the changes of RTTs. This insensitivity might cause the inefficiency of TCP in sending packets and possibly result in congestion in the routers.

4.2 Simulated Networks

All the simulations are conducted using the NS2 network simulator [10] as a platform. TCP-Reno is adopted.

Simulation Topology

We simulate environments where a bottleneck link is between a premises on the client side and an edge router on the ISP side, as in reality. In fact, such links are among the most cost-sensitive and bandwidth constrained components in the Internet, and remains the most concern of a ISP.

A dumbbell topology used in simulations is depicted in Figure 5. There are two servers, two clients, and three Internet routers. The bottleneck link capacity, in our simulations, is set as 2Mb, when the capacity of the other links is either 10Mb or 100Mb as shown in the figure. $R1$ is a core router, while $R2$ is an edge router and $R3$ is a premises. One way TCP traffic has been used in the simulations. There are two kinds of traffic, one from server $s1$ to client $c1$, the other one from server $s2$ to client $c2$. The propagation delays of these two traffic types are 44ms and 80ms, respectively. The AQM schemes are implemented in router $R2$ to conduct the performance comparison with Droptail. The output queue buffer of router $R2$ is 300 packets. A dropping strategy is used to inform the TCP senders.

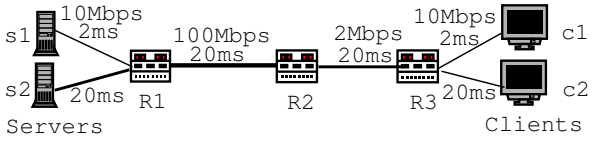

Fig. 5. Network topology

Traffic Pattern

It is well known [11] that Internet traffic tends to be made up of a large number of quite small flows, a small number of very large flows, and nothing much in-between. This observation follows from another well accepted model of Internet traffic – the Poisson Pareto Burst Process (PPBP) model [12, 13].

In this paper, the PPBP traffic model is generated by setting the inter-arrival time of two kinds of traffic subject to non-positive exponential distribution and file sizes of each flow subject to the Pareto distribution. In our simulations, the Pareto distribution has an average flow size of 12 packets (1000 bytes for each packet), and with a shape parameter of 1.2. We vary the inter-arrival time to get different traffic loads to the network. With the topology shown in Figure 5, for example, traffic load is approximatively 80%, 100%, 125% of the bottleneck link capacity, when the inter-arrival time variable λ is 8, 10, 12.5 respectively by using the following formula:

$$traffic_load = \frac{(40 + afs \times 1040) \times 8 \times n}{\lambda \times c} \tag{8}$$

We give the explanation for the above formula as follows. SYN control packets are 40 bytes and each data packet is 1040 bytes with 1000 bytes of data and 40 bytes of header. afs stands for average flow size. Variable n is the number of traffic types, and in our case it is two, one from Server $s1$ to client $c1$, and the other from Server $s2$ to client $c2$. Variable c is the bottleneck link capacity, and we have chosen c as 2Mb.

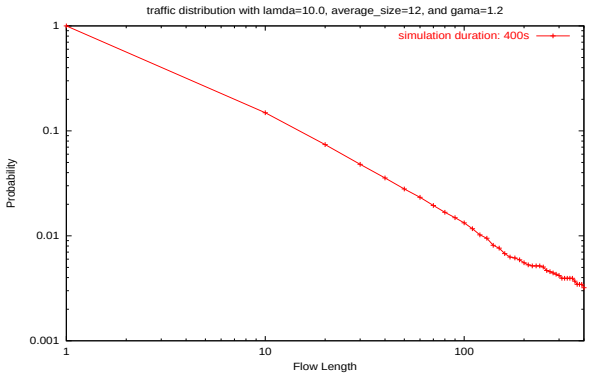

Fig. 6. Traffic pattern with $\lambda = 10$

Before conducting each simulation, the generated traffic based on the method mentioned above has been checked to find a simulation duration long enough to get PPBP traffic. For example, when variable λ is 10, we have found 400s is an appropriate simulation time interval, and the generated traffic is approximately PPBP. Figure 6 shows the traffic pattern using a log scale on the axes.

Simulation Results

The simulations have been carried out with three scenarios, including $\lambda = 8$, $\lambda = 10$, and $\lambda = 12.5$. We randomly choose one TCP connection as a sample to examine the performance of the proposed fuzzy estimator with the comparison of Karn's algorithm in each scenario. The simulation results have been shown in Figure 7, Figure 8, and Figure 9.

- Scenario 1: With the parameter $\lambda = 8$ in the network, the traffic load is 80%, which indicates the network is relatively relaxed, not much congested. In this case, there exists considerable oscillation in RTT caused by the bursty nature of the Internet. The difference between the RTT and RTO is fairly big, though

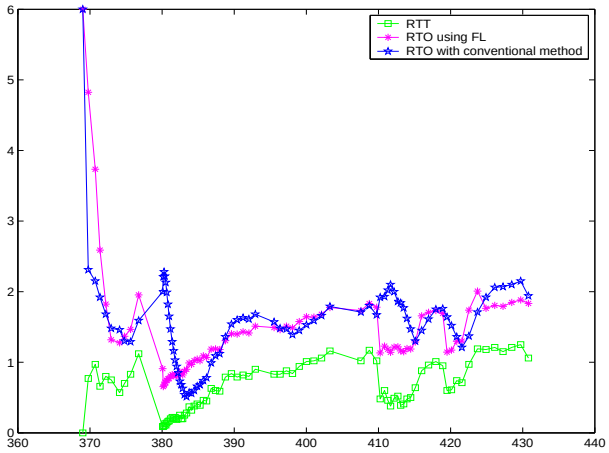

Fig. 7. where $\lambda = 8$

the TCP RTO predicted by Karn's algorithm is converging at the late stage of the experiment.
- Scenario 2: With the parameter $\lambda = 10$ in the network, the traffic load is heavy in the network. In this case, the RTTs are larger than those in the previous scenario, and with smaller oscillation. Like scenario 1, the difference between the RTT and RTO is fairly big, and the RTO predicted by the Karn's algorithm is converging at the late stage of the experiment.
- Scenario 3: With the parameter $\lambda = 12.5$ in the network, the traffic load is very heavy in the network. The output buffer in the bottleneck is full most of the time. And thus, the RTTs keep steady. For a relatively congested network, the RTTs and RTOs predicted by Karn's algorithm are getting closer.

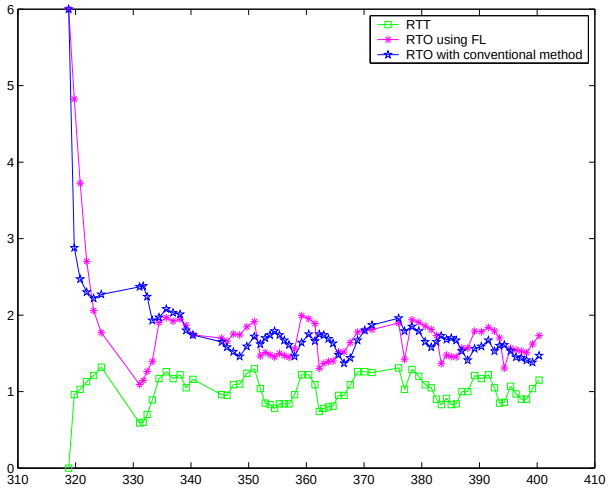

Fig. 8. where $\lambda = 10$

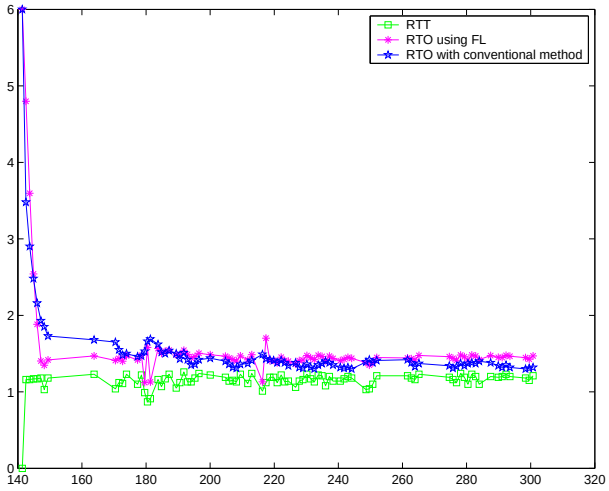

Fig. 9. where $\lambda = 12.5$

All the results indicate that the proposed fuzzy estimator is able to be consistent with the tendency of RTT with a certain desired distance. In addition, for a congested network where the parameter $\lambda = 12.5$, the fuzzy estimator has also reduced TCP timeout, which is obvious at the early phase of the experiment. In the later phase, the fuzzy estimator performed not worse than the TCP-Reno on average, although occasionally the TCP timeout is a little bigger.

5 Conclusion

This paper has presented a fuzzy estimator for TCP timeout and retransmission. The main feature of this method is the application of fuzzy logic prediction. It has shown that the performance of the fuzzy inference system is much finer than the conventional methods. In particular the TimeOut calculated by our fuzzy estimator is always less than that estimated by Karn's algorithm.

The fuzzy estimator for TCP timeout has been also tested on a set of simulated networks with different traffic loads. The fuzzy estimator produced favorable results, although the improvement to the TCP timeout was so obvious as the one on the real network.

This research can be extended in the following directions:

1. The input variables for the fuzzy system can be more specific, which means we can use the RTT directly.
2. The fuzzy rules can be refined using the max-product.
3. This fuzzy logic can be applied in other TCP algorithms such as the slow start, fast recovery and congestion window control.

References

1. Karn, P., and Partidge, C. (1987) *Improving Round-Trip Time Estimates in Reliable Transport Protocols*, Computer Communication Reviews, vol. 17, no. 5 pp2-7 (Aug)
2. Lim, H. H., and Qiu, B. (2001) *Performance Improvement of TCP using Fuzzy Logic Prediction*, Proceedings of 2001 International Symposium on Intelligent Signal Processing and Communication Systems, Nashville, Tennessess, USA, November 20-21, pp152-156.
3. Kosko, Bart, (1992) *Neural Networks and Fuzzy Systems*, Englewood Cliffs, N. J.: Prentice Hall, Inc.
4. Comer, Douglas E. and Stenves, David L.: (1994) *Internetworking with TCP/IP: Design, Implementation and Internals*, vol 3, Prentice-Hall, Inc.
5. Zadeh, L. A: (1973) *Outline of a new approach to the analysis of complex system and decision process*, IEEE Transaction on Systems, Man and Cybernetics SMC-2, pp28-44
6. Zimmermann, H.-J: (1996) *Fuzzy Set Theory and Its Application*, 3rd edition, Kluwer Academic Publishers, Boston
7. Yager, R. and Filev, D. P.: (1994) *Essentials of fuzzy modeling and control*, John Wiley & Sons, Inc.
8. Van Jacobson, Caraig Leres and Steven McCanne: *Freeware tcpdump*, `ftp://ftp.ee.lbl.gov`
9. Larry L. Peterson and Bruce S. Davie (2000) *Computer Networks*, Morgan Kaufman Publishers
10. S. McCanne and S. Floyd, (1997) *Network simulator ns-2*, `http://www.isi.edu/nsman/ns`.
11. V. Paxson and S. Floyd: (1995) *Wide-area traffic: The failure of poisson modeling*, IEEE/ACM Transaction on Networking, 3(3), pp226-244

12. N. Likhanov, B. Tsybakov, and N. D. Georganas: (1995) *Analysis of an ATM buffer with self-similar ("fractal") input traffic*, In Proceedings, IEEE Infocom 1995, pp1-15, April
13. R. G. Addie, T. M. Neame, and M. Zukerman: (2002) *Performance evaluation of a queue fed by a Poisson Pareto burst process*, Computer Networks, 40:377-397, October.