Non-intrusive IP network performance monitoring for TCP flows

B. Ghita, B.M. Lines, S.M. Furnell, E.C. Ifeachor
Department of Communications and Electronic Engineering, University of Plymouth, Plymouth, UK
{b.ghita, sfurnell@jack.see.plym.ac.uk}, {e.ifeachor,b.lines@plymouth.ac.uk}

Abstract: The expansion of the Internet in the past two decades has led to a large amount of traffic being carried over IP (Internet Protocol) networks, most of which is due to web browsing. Unfortunately, the Internet revolution was not accompanied by an improvement in monitoring. Until recently, the main problem that affects TCP (Transmission Control Protocol) performance was considered to be the available bandwidth and, in turn, bandwidth was less of an issue when compared to network availability. This paper presents a method that allows offline, single point, non-intrusive performance measurement for TCP connections. The method avoids all the limitations of present monitoring solutions, i.e. intrusive and / or complex, and offers in-depth information about the performance parameters. This is a first step in defining, evaluating and measuring network Quality of Service for TCP transfers. Test results show that the method is correct for measuring throughput and has an accuracy greater than 95% when determining RTT (Round Trip Time) values, but may have errors of up to 30% when estimating packet loss, due to uncertainty in determining certain events and to differences between various TCP implementations.

I INTRODUCTION

In the last two decades, the unprecedented expansion of the Internet has led to a large amount of traffic being carried over IP networks. According to recent studies on large networks and backbone segments, [1], [2], the majority of Internet traffic is produced by web browsing. Additionally, the content of the web pages has moved from text to multimedia, e.g. images, and even pseudo-real-time traffic, leading to new loss and delay issues. The transport performance of web browsing depends exclusively on the performance of its underlying protocol: TCP. In order to cope with these new requirements, the Internet should be, besides ubiquitous, also fast and ideally loss-free. This is not the case at present, mainly due to the *best effort* character of its core protocol, IP. The first step in improving the current situation would be to evaluate the performance. Unfortunately, until now, little has been done to determine the **quality** of individual Internet TCP connections, which would give the performance of web traffic; current performance measurement methods are either intrusive, or indicate only the overall **quantity** of traffic. Intrusive measurement methods are accurate, but they have several inconveniences: they often require an infrastructure being deployed at the points where test traffic is injected into the network, the measurement is limited to the injected traffic and it is presumed that all the traffic types (e.g.: TCP and ICMP, Internet Control Messaging Protocol) encounter the same behavior from routers.

This paper presents a method that allows single point, non-intrusive performance measurement for TCP connections, together with an implementation which was developed as a proof of concept for this method. The method avoids all the inconveniences of current monitoring solutions, i.e. intrusive and complex, and offers in-depth information about the performance parameters. This is a first step in defining, evaluating and measuring network Quality of Service for TCP transfers.

The rest of the paper is organized as follows: in section II the current state-of-the-art in performance measurement is presented, together with the limitations and disadvantages they include. Section III then describes the underlying mechanisms of TCP. Section IV describes the proposed measurement method, while section V outlines an accompanying proof-of-concept implementation of the method, section VI discusses results of preliminary benchmarking tests and, finally, section VII presents overall conclusions and ideas for future work.

II CURRENT MONITORING EFFORTS

There are three main types of methods to determine the performance of traffic: intrusive, pseudo-non-intrusive, and non-intrusive. The Cooperative Association for Internet Data Analysis (CAIDA) maintains evidence of the efforts related to network monitoring [3]. At present, most of the methods that measure the performance parameters of the network are intrusive. The most commonly used subset of these techniques is based on ICMP messages. They involve two stages: first a probe packet being sent from the monitoring station over the network to a specific target host, then a reply being produced by the target and sent back to the monitoring station. The monitor then determines the parameters of the network by examining timing information of this request / response dialogue. Examples of monitoring tools based on this mechanism are *ping* and *traceroute*, [4]. A more advanced subset of these techniques bases its measurements on TCP transfers, instead of ICMP exchanges, which makes the results equivalent with the real web browsing traffic (as will be described later, TCP behavior is not 'bulk transfer', but governed by certain rules). Examples of network monitors that use such techniques are *pathchar[4]*, *treno* (described in [5]), and *sting* [6].

The second category of monitoring techniques can be termed *pseudo-non-intrusive* methods. They do not

send traffic in order to measure it, but request management information from other hosts to build an image of the network performance. For example, they can interrogate routers about the statistics of the traffic running over the network using SNMP (Simple Network Management Protocol). They are related more to management issues than monitoring itself, as the data is obtained by interrogating databases.

The main advantage of the above categories is that they provide accurate measurement of the focused variables other than probe effect. Unfortunately, both approaches have several disadvantages. The main problem resides in the fact that they inject additional traffic in the network, either to measure it (as intrusive methods do), or to exchange information, which occupies network resources. The two categories also have deployment issues: they require access, to run, to update and to collect data from dedicated software at the 'other' end (i.e. measurement client). Also, in most of the cases, the remote end is inaccessible, therefore a measurement architecture using such methods can easily be brought near failure [7].

Non-intrusive monitoring is the third category of network monitoring methods. The methods included do not send any traffic into the network, but only capture and analyze the traffic transiting the point where the monitoring station is connected. They could represent the perfect solution for continuous monitoring, as they eliminate the disadvantages of previous methods. The main disadvantage is that these methods infer the required parameters from the observed packet flows; their accuracy is strongly related to how the packet exchanges are interpreted.

Unfortunately, the Internet revolution was not accompanied by an improvement in monitoring. Until recently, the main factor that affects TCP performance was considered to be the available bandwidth, as network parameters such as loss and delay were less of an issue than availability and bandwidth. Therefore, the vast majority of the existing non-intrusive monitoring methods are geared towards measuring the overall used bandwidth of a network; the other parameters (loss, delay) can be determined only if the traces are analyzed by a network specialist. The most advanced tools to analyze and interpret TCP traces are the trace analyzers, such as *tcptrace* [8] and *tcpanaly* [9]. but they concentrate more on the TCP behavior than on network performance, therefore they are less suitable for monitoring. Now the problem has changed: broadband access is widely deployed to Internet hosts and the content of the Internet has become multimedia-rich. No matter how much the locally available bandwidth expands, the delay of the packet flows, as they transit the intermediate networks (e.g. satellite links), and loss rate, due to network congestion, remain problematic issues. As a result, the *quality*, i.e. performance, offered to each packet flow not *quantity* should be measured. This paper proposes a technique that evaluates non-intrusively the performance of the traffic transiting the network as observed from a single-point.

III TCP MECHANISMS

The performance of TCP transfers is determined by the download speed. Being a reliable protocol, TCP provides for loss recovery and in-sequence data delivery. In order to perform these functions, the TCP specification [10] includes mechanisms for data ordering, acknowledging and retransmission.

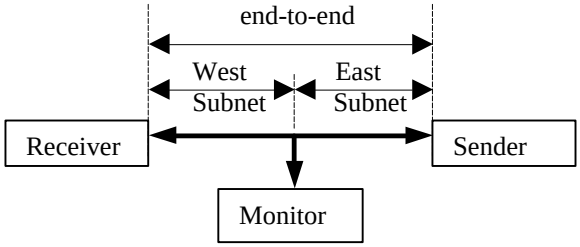
When a TCP transfer takes place, two unidirectional packet flows are established: a download flow, from the sender to the receiver, which carries the actual data transferred, and an acknowledge flow, from receiver to sender, which confirms the data segment. The acknowledge flow allows the sender to determine when / if a data segment arrived at the receiver, and to retransmit it if necessary. There are two indications of packet loss [11]:

- double acknowledges - the receiver confirms repeatedly older data segments; newer packets arrive at the receiver, but there is one of them missing, and the receiver is requesting that one; if the number of double-acknowledges for a segment is higher than a threshold, the segment is re-sent;
- timeouts – a mechanism that estimates the average and deviation of RTT exists within TCP clients, based on the time when data is being sent / acknowledged; if no acknowledge is received in a time higher than the timeout estimate, the data segment is re-sent.

TCP, besides being a reliable protocol, is self-tuning: it evaluates the network status, based on the delay and loss characteristics of the acknowledgement packets, and adjusts its transmission rate accordingly [11]. The sender cannot transmit all the available data, but only segments up to an adjustable congestion window. Events in the network negatively affect the dimension of the congestion window, e.g. a lost packet halves it. Under these conditions, even if bandwidth is sufficient, if a high delay occurs (i.e. it takes a long time for the acknowledges to reach the sender) or a congested network is transited (i.e. increased packet loss), the TCP speed will be reduced radically.

IV MEASUREMENT METHOD

The aim of the proposed method is to evaluate the network performance parameters of the TCP connections. The monitor is positioned somewhere along the path which is transited by the packets. If we consider



the server/client character of the web traffic, we can divide the end-to-end path of the flow into two virtual segments, called *West* and *East*, as in Figure 1.

Figure 1 - Measurement for a sender-receiver configuration

The TCP monitoring process has three steps:

1. Capture the packets (the input).
2. Divide overall flow onto connections, using the IP addresses and transport ports, perform per-flow analysis;
3. Determine the performance parameters (the output).

A list of general network parameters includes: one-way delay, one-way delay jitter, packet loss, and throughput. Due to the characteristics of the measurement, which is single point, there is no method to determine one-way delay information; the delay and delay jitter have to be replaced by RTT and RTT jitter.

The TCP monitor is built similarly to a TCP end-client. The theory behind the monitor was based on the TCP standard and its enhancements, thoroughly described in [10], as well as on the 4.x BSD (Berkeley Software Design) TCP/IP stack implementation, presented in [12]. It is a state machine which has as inputs the packet arrivals and emulates the processing of packets being 'sent' and 'received' as happens within the sender or the receiver part of the TCP client. The only outputs it produces are the obtained performance parameters. The monitor is different from the TCP end-client for several reasons, mainly the inputs of a TCP end client, which include user calls, packet arrivals and time-outs. Because, at the monitoring point, there is no access to the user-TCP interaction, there is no access to user calls and no information about when the timers are set/expire at the endpoint. The states from the original TCP diagram were maintained, but the transition triggers were modified in order to adapt to these unknowns. The transitions in the monitor follow the transitions that happen at the endpoints and they are due to: packet arrival from the endpoint (most of them), specific transition of the corresponding endpoint (for the transitions which have no outputs, and a packet arrival or a user call as an input), or unconditional (due to expiry of a timer – different implementations might have different settings for the timer). In addition to the TCP transition diagram, there were added two additional state machines:

- sender: NO_DATA (the sender received acknowledgements for all the data segments sent) and WAIT_ACK (the sender previously sent data which has not been acknowledged yet)
- receiver: NORMAL (the receiver is acknowledging data), DUPLICATE_ERR (the receiver sent the same acknowledgement twice), DUPLICATE_ERR1 (the receiver sent the same acknowledgement three times)

The sender machine indicates whether or not the sender is idle, while the receiver machine flags packet losses advertised by the receiver.

The main functioning principle is that the TCP monitor emulates the TCP client. Therefore, the monitor maintains relevant information about :

- connection variables (e.g. connection state, sequencing information);
- current values for performance parameters of that specific flow, which is accessed / modified each time when a packet is received;
- information related to past behavior: a memory of 'skipped' segments which contains all the apparently skipped segments, i.e. segments older than current sequence number which have not been passed by the monitoring point yet.

Parameter update depends on the packet status (whether or not it is a 'good for update' packet, or not). By comparing the sequence/acknowledgement number in the TCP header of the packet with the sequence variables of the flow to which it belongs, determines a variable called *PacketStatus*, which defines the data segment within the packet; in parallel, the acknowledgement number informs the receiver part of the flow about the status of data sent.

Several categories are defined to describe data segments, depending on their status:

- *correct* = segment of data in sequence, following last sent segment
- *future* = out-of-order data; the sequence number of the packet is higher than the expected sequence number
- *retransmitted* = old data segment which was transmitted at some moment in the past, and now is retransmitted, probably due to a packet being lost
- *inverted* = old data segment which was misordered (followed a future data segment, but it is only out-of order, not retransmitted).

In addition, two types of acknowledgements are defined:

- *correct ACK* - there is no data to be acknowledged, or the acknowledgement number acknowledges the last transmitted segment);
- *duplicate ACK* - the sender still has unacknowledged data and the acknowledgement number does not acknowledge highest sequence number sent.

The steps of the data analysis are as follows:

1. Determine if the ACK in the packet is *correct*.
2. Determine what type of data is inside the packet.
3. Update the flow variables, depending on the data contained by the packet (if packet is not empty).
4. Separate out-of-order packets from lost-before-monitor packets within the *inverted data* category.
5. Calculate RTT; update the RTT average and jitter.

For a Sender-to-Receiver flow, as pictured in Figure 1, the output parameters of the method are:

- lost packets – two variables: packets lost before the monitoring point and packets lost after the monitoring point;
- out-of-order packets;
- total number of transmitted packets (including retransmissions);
- RTT average;
- RTT jitter;
- useful data throughput – related to the amount of valid data that was received;
- total data throughput – related to the total amount of data that was sent to the receiver (including retransmissions).

The data throughput measurements have 100% accuracy, as they are obtained by subtracting last transmitted sequence number and initial sequence number.

The implementation consists of a program written in C++, which captures the packets, parses the packet

headers and identifies the fields of interest, identifies the flow to which the packet belongs, based on the IP addresses and ports fields within the IP and TCP headers, performs the analysis, and displays the result in a text form. It is not the purpose of this paper to discuss the characteristics of the software program itself, but, during the development phase, relevant issues were raised related to TCP monitoring.

V ERROR SOURCES

Two main sources of errors were observed:

- limitations due to the monitor position, which made some of the packet loss events invisible and affected RTT measurements;
- differences within the variety of TCP implementation that exist, which made packet loss, timeout and congestion window estimations inaccurate.

The monitor is positioned, as pictured in Figure 1, somewhere along the path transited by the packet flow. Because of this, the packets being exchanged by the two endpoints allow RTT measurement only for the East network, because no (or little) data is sent from the receiver to the sender, and TCP does not perform acknowledgement-of-acknowledgement. This affects the measurement dramatically if the monitor is 'near' the receiver, as no significant RTT measurements are possible in this case.

Differences between TCP implementations were identified to create large changes in TCP behavior in earlier studies, such as [9]. The method proposed was to profile / identify each type of implementation. This approach is valid at a certain time, but it has to be renewed later, when new implementations, with different behaviors, are released. The differences between implementations are part of the TCP philosophy 'be conservative in what you do, be liberal in what you accept from others' [11], and the endpoints can adapt to it, but it removes the packet arrival patterns which exist within observed data transfer. Therefore, this method aims to identify as accurately as possible the events that produce the behavior of the monitored TCP transfers, while maintaining a generic aspect.

Initially, the method included a mechanism to follow the congestion window evolution at the sender. Unfortunately, the TCP client from Windows 98 had a strange evolution, even under no-impairments conditions: instead of being maintained high, the congestion window was exponentially raised from 1 segment to a maximum of 4 segments, according to the acknowledgements, then it was reset to a single segment. It is difficult to determine if this was a bug in that specific implementation or a congestion window limitation.

There were also differences between the Windows 98 and Linux TCP implementations. Linux implementation used SACKs (selective acknowledgements): if a single packet is lost, in the middle of a congestion window, the Linux TCP sender transmits the packet, adjusts the congestion window, then continues (i.e. waits for the next acknowledgement). In contrast, the Windows 98 TCP implementation did not implement this feature, therefore retransmitted the entire remaining window; this introduces an error factor in the monitoring method, as the TCP client retransmits packets which were not lost.

Timeout estimation is another feature that proved to be unusable because of both of the mentioned categories. If variance occurs and cannot be detected (such as variance in the RTT measurement for the East segment, as described above) the estimated timeout of the monitor is different from the one of the sender. The timeout is a binary decision: if an acknowledgement does not arrive in time, the packet is considered lost. This type of decision requires a very fine granularity of the measurement, which cannot be achieved under such configuration. This limitation affects also the packet loss estimation: lost packets due to a timeout events cannot be observed, therefore the measured value is different from the real one.

In spite of its ability to provide important data, acknowledgement interpretation was not used. The decision was taken due to two factors: implementations differences (acknowledging policy, e.g. delayed acknowledging, depends on the receiver TCP implementation) and reverse path packet loss (while lost data packets are retransmitted, lost acknowledges are not).

VI VALIDATION TESTS

The method was continuously benchmarked using a Ethernet network testbed, pictured in Figure 2, which emulates various network conditions, using the facilities of a network emulation tool, *NISTNet* [13].

For transfers, the two station used, alternatively, two types of TCP implementation: Windows 98 and Linux OS, each of them with its own TCP implementation. The two links from Figure 2 are, in fact, two routers with *NISTNet [13]* installed on them, which delay and / or discard packets according to the rules specified by the user. As the monitoring station was placed in the middle, Figure 2 can be mapped onto Figure 1: Link 1 is West Subnetwork, and Link 2 is East Subnetwork.

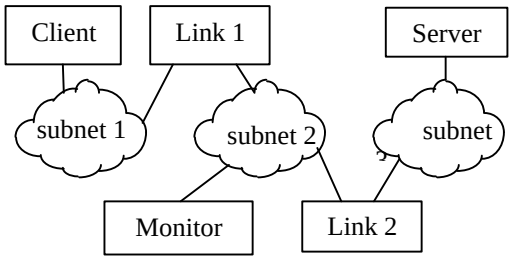


Figure 2 Test configuration

First, preliminary tests were performed to determine the propagation times of the testbed when no impairments are introduced using *NISTNet*. Two simplifying hypothesis were introduced:

- the data packets (from Sender to Receiver) are all full (1518 bytes frames) and the acknowledges (from Receiver to Sender) are all empty (64-bytes frames); therefore the RTT can be computed as:

$$RTT\big|_{end-to-end} = \frac{RTT\big|_{1518bytes} + RTT\big|_{64bytes}}{2} \quad (1)$$

- the network is symmetric (the two subnetworks, West and East, have the same properties):

$$RTT\big|_{West} = RTT\big|_{East} = \frac{RTT\big|_{end-to-end}}{2} \quad (2)$$

The tests consisted of sending batches of 100 ping packets with frame size of 1518 bytes (full data frames) between the two stations, with delays between packets that correspond to a throughputs between 10 kB/s and 500 kB/s. The test was then repeated for 64-byte frames (empty acknowledgement frames). From the preliminary tests it was concluded that, without introducing any degradation, the following approximate values can be considered:

- end-to-end RTT for the network testbed is 10ms
- end-to-end RTT jitter is 0.1ms

Three tests were performed, in order to determine the accuracy of the method. A large file (2.6 MB) was chosen to be transmitted in order to: eliminate the transient effects from the beginning (i.e. slow start) and obtain more accurate average values. The transfer was realized via FTP (the application layer does not influence TCP behavior). The tests consisted of setting certain values for delay or jitter, making the transfer and monitor it using the method implementation, then read the results.

1. Test 1

Conditions of testing: no degradation

Results of measurement:

- RTT average = 7.17 ms
- RTT jitter = 6.65 ms
- RTT packets = 48
- lost packets east / west = 0 / 61
- number of data packets = 1009
- inverted packets = 0

Conclusion: the RTT and jitter measurement values have the same order with previous findings, based on ping measurements; the aim is not to obtain a higher precision for these values, because the area is near the actual propagation and processing time required under no network degradation.

Observations (will be detailed in next test):

- the number of reported lost packets is non-zero, although no lost actually occurred;
- the number of RTT packets is low.

2. Test 2

Conditions of testing: constant delay, variable deviation, no packet loss; delays introduced:

- R1(B $\rightarrow$A) = 400 ms;
- R1(A $\rightarrow$B) = 300 ms;
- R2(B $\rightarrow$A) = 200 ms;
- R2(A $\rightarrow$B) = 100 ms.

The measurable RTT value is:

$$RTT = R1_{B\rightarrow A} + R2_{A\rightarrow B} + R \ = 710ms \quad (3)$$

The measurement results are presented in Table 1:

Jitter introduced	0	10	50	100
Jitter measured	21.2	48.5	167.8	96.4
RTT measured	728.4	741.2	705.3	695.0
RTT error	2.62	4.45	-0.6	-2.14
RTT packets	17	32	15	14
Lost pkts (East)	0	0	0	0
Lost pkts (West)	1	125	230	250

Units: jitter [ms], error [%]

Table 1 - Delay and jitter measurement results

Conclusions:

- RTT is estimated correctly (i.e. with less than 5% error) in most of the situations;
- RTT measured jitter varies from the introduced jitter – this metric is differently measured from the value produced by *NISTNet*; see also observations below.

Observations:

- the number of RTT measurement packets is relatively low (the total number of data packets was around 2000 packets). This is because RTT is determined only for ACK arrived for the last data packet seen, i.e. acknowledgements of entire packets / congestion windows. This makes the jitter value, as measured by the method, different from the jitter introduced by *NISTNet*;
- although no packets are lost, the number of packets presumed lost in West subnetwork is non-zero, and it is actually increasing up to 10% of the number of transmitted data packets. As only the losses for B$\rightarrow$A are measured, this is actually the number of 'lost-after' packets. The TCP analysis method considers 'lost-after' all packet retransmissions visible to the monitor. These retransmissions are not actually due to packet losses, but due to TCP erroneous retransmission timeout (RTO) estimation at sender, i.e. sender does not receive a confirmation for the packet in a time lower than the estimated RTO and redundantly sends the segment again.

It must be said that, although the erroneous RTO calculation events are not due to 'lost packet', they have the same impact on the transfer as a lost packet:

- the sender adjusts its congestion window as if a timeout occurred;
- the bandwidth is additionally loaded;
- the second reception of the packet is ignored at the receiver (the segment is discarded) – the transmission is useless.

3. Test 3

Conditions of testing: no delay or jitter introduced; variable, symmetric loss, set at 1%, 2% and 5%. The tests for losses higher than 5% failed, because the TCP connection timed out – the TCP sender retransmits the same segment a number of times, if loss was due to timeout, then gives up and closes the connection.

NISTNet maintains the number of lost packets, facility which was very useful in this case, as it allows comparison between the reported number of discarded packets (by *NISTNet*), and the measured (estimated) number of lost packets, as determined by the method. In Table 2 summarizes the conditions of the test by the

packet loss *set* and *reported* columns, and estimation results by the *measurement* column. A comparison between the estimated and the reported values is made in the *Error* column.

Subnet	Packet loss			Error [%]
	set [%]	reported [pkts]	measured [pkts]	
East	1	24	21	12.5
West	1	23	30	30.4
East	2	40	36	10.0
West	2	39	43	10.2
East	5	102	90	11.7
West	5	112	116	3.57

Table 2 - Packet loss measurements

4. Conclusions:

The differences between the values introduced and the ones measured are very high. In all the cases, the measured loss for East Subnetwork (packets *lost before* the monitor – between the Sender and the monitor) is lower than the loss introduced, while for the West Subnetwork, (packets *lost after* the monitor – between the monitor and the Receiver) the measured loss is higher than the loss introduced.

The differences for the 'lost after' category result from differences between TCP behavior: The monitor cannot determine whether the packets were lost or not, so it considers them all lost and retransmitted. This also applies for 'lost after' category.

The differences for the 'lost before' category result from following situations:

- if there is a multiple loss (a packet is lost repeatedly more than once) before the monitor, the monitor can identify only a single loss;
- if a timeout occurs due to the last packet in a transmission window being lost before the monitor, the sender retransmits the packet. Still, no apparent inversion can be detected, because the sender did not transmit any other packets between the two transmissions of the timed-out packet.

VII CONCLUSIONS AND FUTURE WORK

The article presented a method to determine non-intrusively the performance parameters of individual TCP connections. The method represents an important contribution to the network monitoring area, as current methods are intrusive, therefore create additional traffic, and require different degrees of cooperation from the far end.

The validation tests evaluated the accuracy of the method; they were performed in a controlled environment, using two Windows 98 / Linux TCP clients which exchanged data via a TCP connection over an emulated network (*NISTNet*). They proved that the method is exact for measuring throughput and has an accuracy higher than 95% when determining RTT values. Unfortunately, errors up to 30% can appear when measuring packet loss. These errors are due to uncertainty in determining certain events, such as timeout, and to differences between various TCP implementations. Several possible enhancements within the method that would allow a better understanding of the TCP behavior had to be suspended because of the identified differences.

Future work will concentrate mainly on improving the estimation of packet loss. A first step will be to produce an estimate of the timeout, based on an intelligent analysis method, such as fuzzy logic, of the connection *a posteriori*; this approach would produce a better estimate of packet loss. The next step will be to determine the relationship between throughput, as an overall measure of the performance, and packet loss and delay, as performance parameters, adjustable from the management point of view.

Acknowledgments: We are grateful to Acterna for supporting this work.

REFERENCES

[1] Thompson K., Miller G.J., Wilder R., 'Wide-Area Internet Traffic Patterns and Characteristics', in *IEEE network*, nov-dec 1997

[2] Hwang A., 'Observation of Network Traffic Patterns at an End Network: Harvard University', BA Thesis, Harvard college, April 1998

[3] CAIDA, 'The Cooperative Association for Internet Data Analysis website', http://www.caida.org

[4] Jacobson V., Paxson V., 'LBNL's Network Research Group homepage', http://www-nrg.ee.lbl.gov

[5] Mathis M., Mahdavi J., 'Diagnosing Internet Congestion with a Transport Layer performance Tool', in *Proceedings of INET '96*, June 1996

[6] Savage S., 'Sting: a TCP-based Network Measurement Tool, in *Proceedings of the 1999 USENIX Symposium on Internet Technologies and Systems*, October 1999

[7] Paxson V. et al, 'Experiences with NIMI', in *Proceedings of Pasive and Active Measurement,* April 2000

[8] Ostermann S., 'tcptrace home page', http://www.tcptrace.org

[9] Paxson V., 'Automated Packet Trace Analysis of TCP Implementations', in Proceedings of SIGCOMM '97, September 1997

[10] DARPA, 'Transmission Control Protocol – RFC 793', September 1981

[11] Stevens W., Wright G., 'TCP/IP Illustrated. Vol 1 – The Protocols', Adison-Wesley, 1994

[12] Stevens W., Wright G., 'TCP/IP Illustrated. Vol 2 – The Implementation', Adison-Wesley, 1995

[13] Carson M., "NISTNet network emulator homepage", http://snad.ncsl.nist.gov/itg/nistnet/, 2001